НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

«На правах рукопису»
УДК __004.42, 004.56__

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Система захищеного та фрагментарного виконання досліджень»

Виконав:
студент VI курсу, групи ІП-94мп
Новак Володимир Юрійович _____

Керівник:
доцент, кандидат технічних наук,
Волокита Артем Миколайович _____

Рецензент:
доцент кафедри АУТС, кандидат технічних наук,
Катін Павло Юрійович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *Обчислювальної техніки*
(повна назва)

Освітньо-кваліфікаційний ступінь *магістр*
(назва ОКР)

Спеціальність *121. Інженерія програмного забезпечення*
(код і назва)

Спеціалізація *121. Інженерія програмного забезпечення комп'ютерних систем*

ЗАТВЕРДЖУЮ
Завідувач кафедри
Стіренко С.Г.
(підпис) (ініціали, прізвище)
« » 2020 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Новаку Володимиру Юрійовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації Система захищеного та фрагментарного виконання досліджень

Науковий керівник дисертації доц., к.т.н., Волокита Артем Миколайович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від ***« 26 » 10 2020 р. № 3132-с***

2. Строк подання студентом дисертації 25.11.2020

3. Об'єкт дослідження Процес захисту продажу та виконання досліджень, які можуть бути розділеними на частини

4. Предмет дослідження Методи та засоби розробки захищеної системи для продажу досліджень: механізми аутентифікації і авторизації, захищене збереження результатів досліджень, алгоритми шифрування та хешування.

5. Перелік завдань, які потрібно розробити:

- Аналіз існуючих рішень, які дозволяють проводити продаж та виконання досліджень.
- Розробити рішення для продажу різноманітних досліджень.
- Забезпечити захист розробленого рішення, вказати у записці до наукової роботи методи та алгоритми, які були для цього використані.
- Забезпечити механізм розбиття досліджень на частини.
- Забезпечити можливість розбиття оплати за різними частинами досліджень.
- Надати користувачам можливість для дискусій та обговорення цін, умов та процесу виконання замовлень.

6. Консультанти розділів дисертації:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Жабін В. І.		

7. Дата видачі завдання 18.12.2019

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
/.	*Затвердження теми роботи*	*10.04.2020-14.05.2020*	
2.	*Вивчення та аналіз завдання*	*15.05.2020-14.06.2020*	
3.	*Розробка архітектури та загальної структури додатку*	*15.06. 2020-10.09.2020*	
4.	*Програмна реалізація додатку*	*10.09. 2020-09.10. 2020*	
5.	*Оформлення пояснювальної записки*	*10.10. 2020-23.11. 2020*	
6.	*Передзахист*	*25.11.2020*	
7.	*Захист*	*17.12.2020*	

Студент _____ _____
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ _____
(підпис) (ініціали, прізвище)

РЕФЕРАТ

на магістерську дисертацію

виконану на тему: Система захищеного та фрагментарного виконання досліджень

студентом: Новаком Володимиром Юрійовичем

Дипломна робота другого (магістерського) рівня вищої освіти на тему «Система захищеного та фрагментарного виконання досліджень» складається зі вступу та 5 розділів. Загальний обсяг роботи: 109 сторінок, 42 таблиці, 31 рисунок, 3 додатки. Перелік посилань нараховує 30 найменувань.

Актуальність. Актуальність теми дослідження зумовлено необхідністю пошуку нових та зручніших видів продажу інтелектуальних послуг, для спрощення надходження коштів та можливості безпосередньо впливати на хід виконання роботи на різних етапах.

Зв'язок роботи з науковими програмами, планами, темами. Дипломну роботу (магістерського) рівня вищої освіти було виконано в Національному технічному університеті України «Київський політехнічний інститут імені Ігоря Сікорського» відповідно до планів науково-дослідних робіт кафедри обчислювальної техніки.

Метою і завдання дослідження. Метою магістерської роботи є забезпечення ефективного, зручного та захищеного механізму продажу досліджень, з можливістю декомпозиції завдання на частини.

Для досягнення мети дослідження поставлено і вирішено такі завдання:

- Аналіз існуючих рішень, які дозволяють проводити продаж та виконання досліджень.
- Розробити рішення для продажу різноманітних досліджень.

- Забезпечити захист розробленого рішення, вказати у записці до наукової роботи методи та алгоритми, які були для цього використані.
- Забезпечити механізм розбиття досліджень на частини.
- Забезпечити можливість розбиття оплати за різними частинами досліджень.
- Надати користувачам можливість для дискусій та обговорення цін, умов та процесу виконання замовлень.

Об’єкт дослідження - процес захисту продажу та виконання досліджень, які можуть бути розділеними на частини.

Предмет дослідження - методи та засоби розробки захищеної системи для продажу досліджень: механізми аутентифікації і авторизації, захищене збереження результатів досліджень, алгоритми шифрування та хешування.

Наукова новизна одержаних результатів - запропоновано спосіб захищеного продажу досліджень, який відрізняється від існуючих розбиттям досліджень на частини, та блокуванням можливості несанкціонованих дій з боку учасників процесу, що дозволяє організувати контрольований з обох сторін процес продажу та виконання досліджень.

Практична цінність. Запропоновані заходи удосконалення захищеності додатків можуть бути впроваджені численними спеціалістами, для підвищення та забезпечення належного рівня безпеки у існуючих рішеннях або застосувати їх для розробки нових продуктів.

Ключові слова

Захист даних, продаж досліджень, IPFS, блокчейн, обфускація.

ABSTRACT

for master's degree dissertation

made on the theme: "Protected and fragmentary research system"

by student: Novak Volodymyr Yuriyovych

Diploma work for the second (master's degree) level of the higher education on the theme « Protected and fragmentary research system » consists of introduction and 5 parts. Total workload: 109 pages, 42 tables, 31 images, 3 additions. The list of references contains of 30 items.

Topic Relevance. Topic relevance is caused by the necessity of finding the new and more comfortable ways of selling of intellectual services, simplification of withdrawing of funds and ability to directly control the flow of doing the work on different stages.

Thesis connection to scientific programs, plans, and topics. The thesis was prepared according to the scientific research plan of Computing Engineering Department of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute.".

Research goal and objectives. The goal of the thesis is providing of effective, comfortable and secured way of selling of researches with ability to decompose them into the parts.

For reaching the goal the next tasks were set and completed:

- Analysis of the existing solutions, that allow users to sell and complete the researches.
- Development of the solution for selling of researches of different types.
- Provide the security for the solution, write in the thesis for the research methods and algorithms that were used.

- Provide the decomposition of the researches into parts.
- Provide the ability for paying for different research parts.
- Give the ability to discuss the prices and the terms and process of doing the orders.

Object of research – process of securing the selling and completion of researches, that can be decomposed into parts.

Subject of research – methods and tools for developing the secured system for selling of researches: authentication and authorization mechanisms, secured saving of results, encryption and hashing algorithms.

Scientific contribution – proposed the way for securing the selling of researches, that differs from the existing by ability to be decomposed into parts and blocking the unauthorized actions by participants in the process, that allows to provide the controlled from the both sides process of selling and completing the researches.

Practical value. Proposed ways of improving the security of the applications that can be used by different specialists for improving and providing the higher level of security in existing or new products.

Keywords

Data protection, selling of researches, IPFS, blockchain, obfuscation.

ЗМІСТ

ВСТУП

Економіка в наш час відіграє надзвичайно важливу роль у нашому житті. Вона є двигуном життя – саме вона рухає сучасним суспільством, визначаючи ціну певних речей та товарів, даючи мільярдам людей роботу, ціль і надію на завтрашній день.

Економічні зв’язки значно глобалізували світ, а з поширенням всесвітньої мережі Інтернет – це тільки збільшилось [1]. В наш час, інтернет став зручним місцем для заробітку та віддаленої роботи. Так як пропозицій досить багато, то і людей, які можуть виконати певне дослідження або задачу також немало.

Ріст мережі Інтернет, а також її поширення, з ним і перенесення багатьох послуг в режим «онлайн», здійснили неймовірний ріст довіри – інколи не будучи навіть безпосередньо знайомими з виконавцем, замовники почали довіряти їм виконання різного роду завдань на відстані.

Якщо 15 років тому компанія виходила в мережу виключно для підтримки свого престижу, то сьогодні – це як мінімум ще один крупний канал продажу товарів та послуг. У часи кризи 2008-2009 років підприємства традиційного сектора торгівлі були змушені скоротити штат і обсяги продажів, а інтернет-сектор навпаки збільшив торговий оборот за цей період на 46%.

Через десятиліття перед людством постав ще більш суворий виклик – коронавірус. При загостренні пандемії, звичні контакти в офісах, магазинах та навіть на вулиці стали небезпечними. Безумовно, дані обставини залишать глибокий слід у способі життя більшості людей, а для деяких секторів економіки в часи повного локдауну Інтернет залишається єдиним каналом збуту товарів та послуг.

В наш час, існують сотні платформ для виконання широкого спектру задач – від математичних завдань з обрахунку певних складних величин для великих підприємств і компанії, до творчих, які потребують написання художніх текстів, розробки дизайну веб-сайтів або створення 3D-моделей для ігор. Також популярними є замовлення для здійснення наукових досліджень,

соціологічних опитувань, тощо. Більшість цих послуг є, звичайно, платними. Хоча є такі сфери, як програмування, наприклад, де розробники діляться своїми вдалими розробками - ними можуть користуватись усі охочі. Проте, варто відмітити, що більшість великих проектів, які застосовуються корпораціями і звичайними, рядовими компаніями підтримуються за рахунок внесків.

Виконання завдань та досліджень через Інтернет має доволі великий список переваг [2]:

- Вільний графік роботи
- Оренда офісу не потрібна
- Не потрібно витрачати час на поїздку на роботу
- Дохід не обмежений ставкою
- Більший спектр вибору фахівців

Для покращення існуючих рішень та підвищення ефективності як виконавців, так і замовників, було вирішено розробити систему, яка б дозволяла продавати різноманітні дослідження, з можливістю декомпозиції завдання на частини, з забезпеченням оплати за кожну зі зроблених частин.

Актуальність теми дослідження зумовлено необхідністю пошуку нових та зручніших видів продажу інтелектуальних послуг, для спрощення надходження коштів та можливості безпосередньо впливати на хід виконання роботи на різних етапах.

Метою роботи є забезпечення ефективного, зручного та захищеного механізму продажу досліджень, з можливістю декомпозиції завдання на частини.

Перед виконавцем роботи було поставлено наступні завдання для дослідження:

- Розробити рішення для продажу різноманітних досліджень.
- Забезпечити захист та надійність розробленого рішення, вказати у записці до наукової роботи методи та алгоритми, які були для цього використані, обґрунтувати їх вибір.
- Забезпечити механізм розбиття досліджень на частини.

- Забезпечити можливість розбиття оплати за різними частинами досліджень.
- Надати користувачам можливість для дискусій та обговорення цін, умов та процесу виконання замовлень.

РОЗДІЛ 1.

ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Перед початком виконання дипломної роботи було проведене детальне дослідження предметної області та існуючих рішень. У нашу пору, є дуже багато програмного забезпечення, електронних ресурсів та застосунків, які певним чином пов’язані з продажом різноманітних досліджень. Розглянемо існуючі аналоги відповідно до кожної з категорій досліджень:

- Freelancer - міжнародний сайт для виконання завдань на різноманітну тематику
- Upwork – сайт для пошуку регулярної та нерегулярної роботи в мережі Internet
- Marketing.rbc.ru - російський ресурс для продажу маркетингових досліджень

1.1. Загальні відомості про Freelancer.com

Freelancer.com [3] – це популярний веб-сайт для фрілансерів, один з конкурентів Upwork (раніше oDesk). Більше 25 мільйонів зареєстрованих користувачів виконали понад 12 мільйонів проектів за допомогою Freelancer.com, що означає, що багато компаній у всьому світі використовують цих незалежних фахівців для розробки програмного забезпечення та розробки програм, написання, дизайну та інших бізнес-послуг.

У своєму огляді на Freelancer.com ми ознайомимося з тим, як замовники та виконавці зараз використовують сайт, а також як його функціонал, зокрема, оплата та відстеження часу, відрізняються від конкурентів.

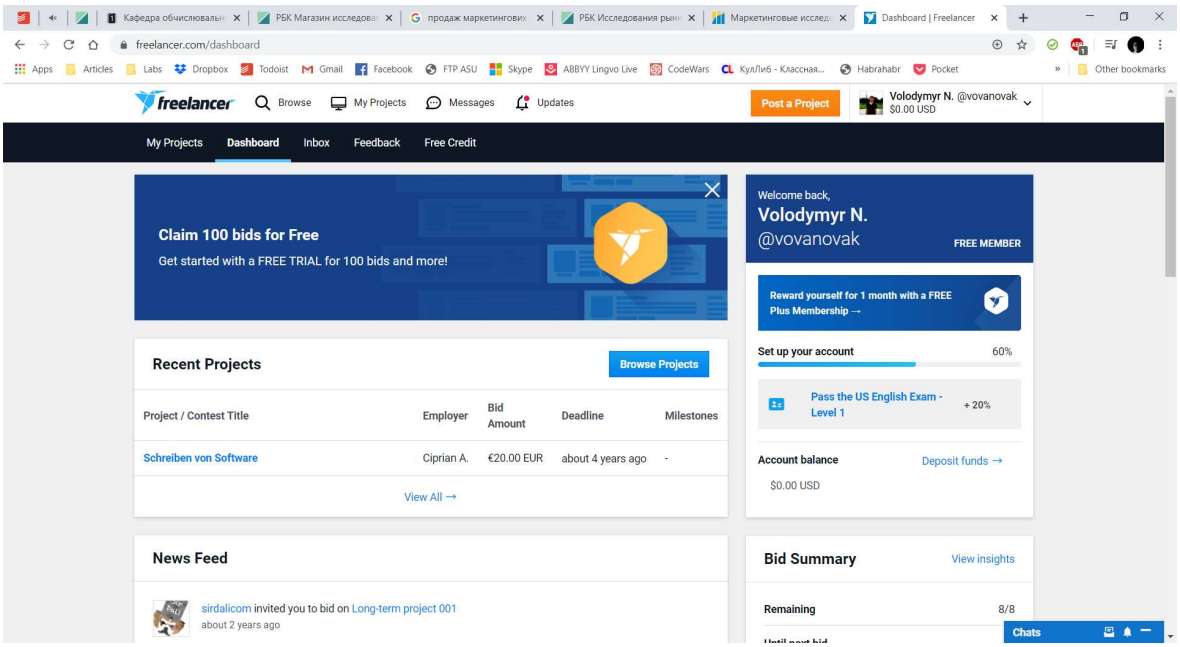


Рис. 1.1. Головна сторінка сайту Freelancer.com

На рис 1.1 можна побачити головну сторінку сайту Freelancer.com. На ній користувач має можливість переглянути свої поточні завдання. Також дана сторінка містить дані про баланс рахунку користувача. Можна переглянути останні оновлення про свої завдання. В додаток вбудовано зручний месенджер для комунікації між користувачами. Варто відзначити, що на сайті користувач не реєструється як замовники або виконавці, тобто в перспективі користувач може як виконувати, так і створювати завдання.

Для операції з транзакціями веб-сайт підтримує наступні платіжні системи:

- PayPal
- Skrill
- Payoneer

Також є можливість виводу коштів напряму на картку, проте це працює лише у певних країнах. Відповідно, для кожної країни в залежності від вибору типу оплати залежить і швидкість виводу грошей з платформи.

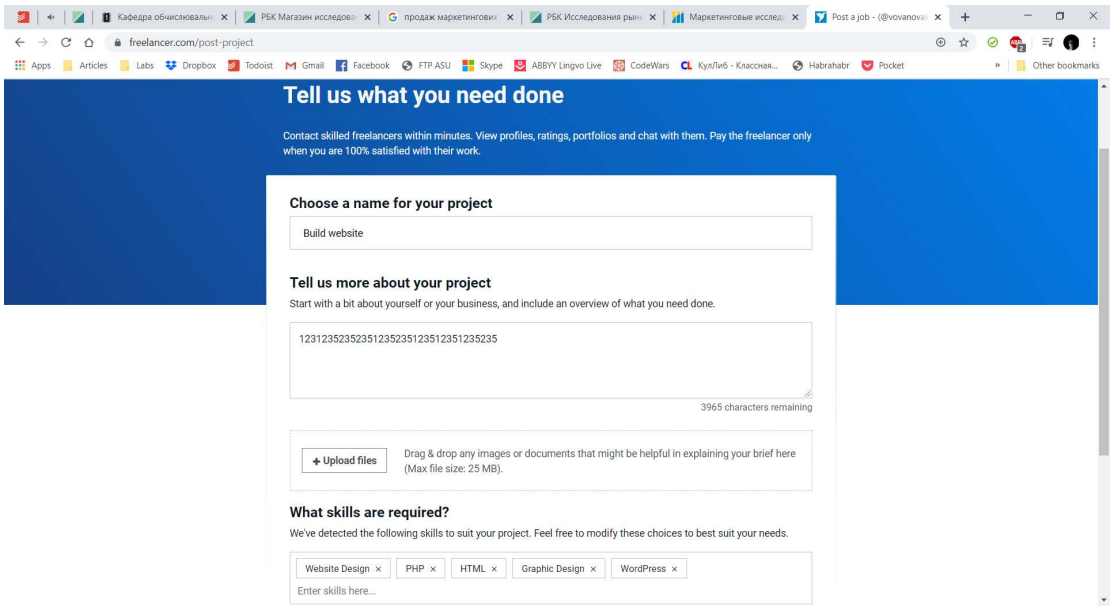


Рис. 1.2. Створення завдання на сайті Freelancer.com

На рис 1.2. замовник має можливість створити завдання. В ньому він вводить необхідний опис, вибирає категорії до якого відноситься нове завдання та вводить суму, яку він згідний виплатити в якості нагороди за виконане завдання.

На даному сайті реалізована функція розділення завдання на певні частини. Тут вони називаються milestones. Це дозволяє в кращій мірі розбити завдання і, по мірі, виконання перевіряти прогрес. Проте, варто відзначити, що виконавець не може надіслати оплату за виконаний частину – він оплачує конкретно виконане замовлення.

Для виконавця це може бути проблемою, так як уже було сказано вище, оплата завдання проходить в різних країнах з різною швидкістю (інколи більше тижня).

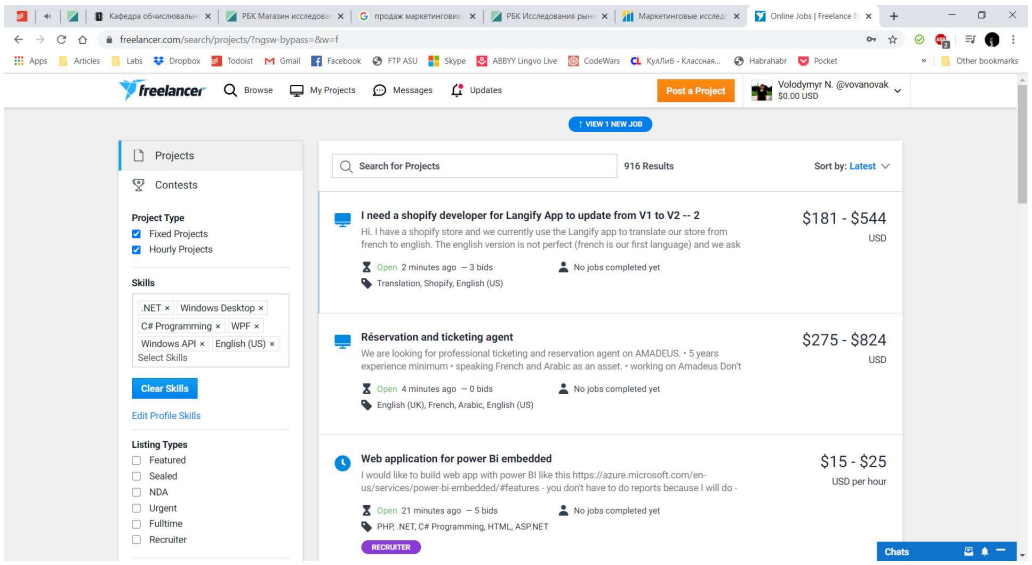


Рис. 1.3. Перегляд завдань до виконання

На рис 1.3. зображено список завдань доступний для всіх користувачів. Потенційні виконавці мають можливість фільтрувати завдання за категоріями, цінами та назвами. При заявці на завдання виконавці мають можливість залишити свою суму та коментар. Відповідно наступним кроком, є вибір виконавця. Після вибору, завдання виконується. Через якийсь час замовник приймає роботу.

З досвіду користування даним ресурсом можу відмітити наступні недоліки:

- Немає роздільної оплати
- Немає гарантії того, що у замовника є необхідна сума для оплати завдання (був неприємний випадок з не оплатою виконаної роботи)
- Відсутня можливість перегляду виконаної частини без згоди виконавця

1.2. Загальні відомості про Upwork

Upwork [3] – веб-платформа, яка надає глобальний майданчик з пошуку роботи й низку програмних продуктів для роботодавців, які хочуть винаймати й керувати віддаленими спеціалістами. Upwork заснували 2003 року грецькі підприємці: Одіссей Цалатос і Стратіс Караманлакіс. На рис. 1.4 зображена головна сторінка користувача на ресурсі Upwork.

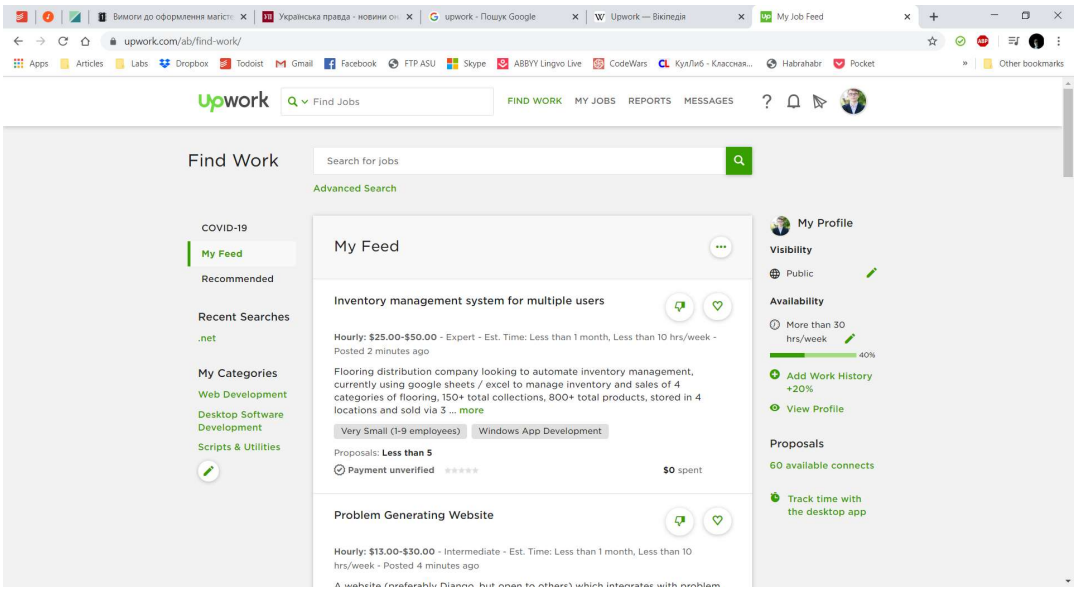


Рис. 1.4. Головна сторінка користувача на Upwork

Upwork дає змогу клієнтам створювати завдання та дослідження для різноманітних команд розробників. Координація й оплата відбуваються за допомогою програмного забезпечення компанії та сайту. Попередня назва компанії, «Одеск» — це скорочення від «онлайн стіл», що відображало намір компанії дозволити будь-кому працювати в будь-якому місці, у будь-який час. Потенційні клієнти можуть створювати проекти безкоштовно, а підрядники можуть створювати профілі та робити пропозиції на виконання проектів.

Компанія збирає 20 відсотків оплати за роботу від роботодавця фрилансеру за перші зароблені $500, 10% за доходи від $500.01 і до $10,000, та 5% за кошти зароблені понад $10000. На додаток до майданчика проектів,

послуг платежів та бухгалтерії компанія пропонує колаборативне програмне забезпечення, «Upwork Team App», що дає змогу клієнтам бачити роботу виконавця в той час, коли він перебуває в режимі оплачуваного часу. Слід зауважити, що виплати здійснюються не миттєво, а лише через 6 днів так званого «безпечного періоду», протягом яких виконавець може відмовитись від виплати в разі, якщо буде підозра на несанкціоноване зняття. Також необхідно близько двох днів для виводу коштів на рахунок виконавця.

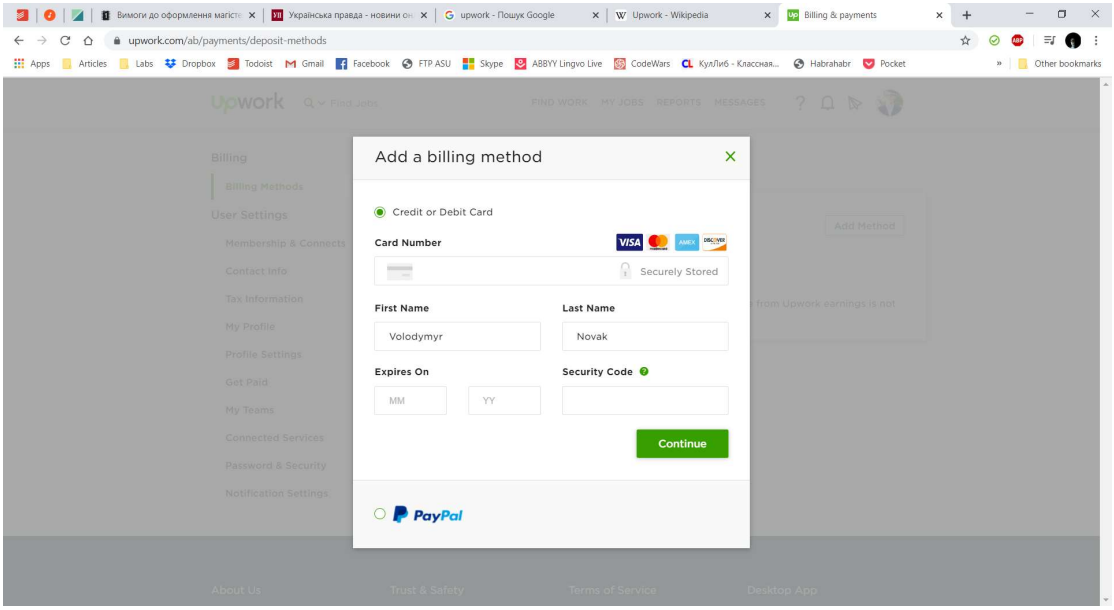


Рис. 1.5. Сторінка додавання способу оплати

На рис. 1.5 зображене вікно проведення платежів. Виводити кошти з Upwork можна за допомогою систем PayPal, Wire Transfer, Direct Deposit/ACH, а також на картки Payoneer та Skrill.

Загалом, Upwork дозволяє клієнтам проводити інтерв'ю, обирати виконавців та взаємодіяти з ними через вищеописану платформу. На сайті реалізований свій механізм чату між користувачами. Також однією з цікавих можливостей обліку платформи порівняно з конкурентами є можливість відстження часу потраченого на виконання, для ефективної роботи механізму почасової оплати.

Серед недоліків сервісу можна виділити наступні пункти:

- Відсутнє регулювання оплати виконання завдання
- Високі комісії
- Висока конкуренція

1.3. Загальні відомості про marketing.rbc.ru

Останнім з обраних аналогів є сайт для продажу маркетингових досліджень marketing.rbc.ru [5]. На даному сайті користувач має можливість замовити маркетингове дослідження на певну тематику. Ресурс підтримує можливість створення персоналізованого дослідження на вимогу замовника. Також є функціонал продажу уже зроблених досліджень за певну визначену суму.

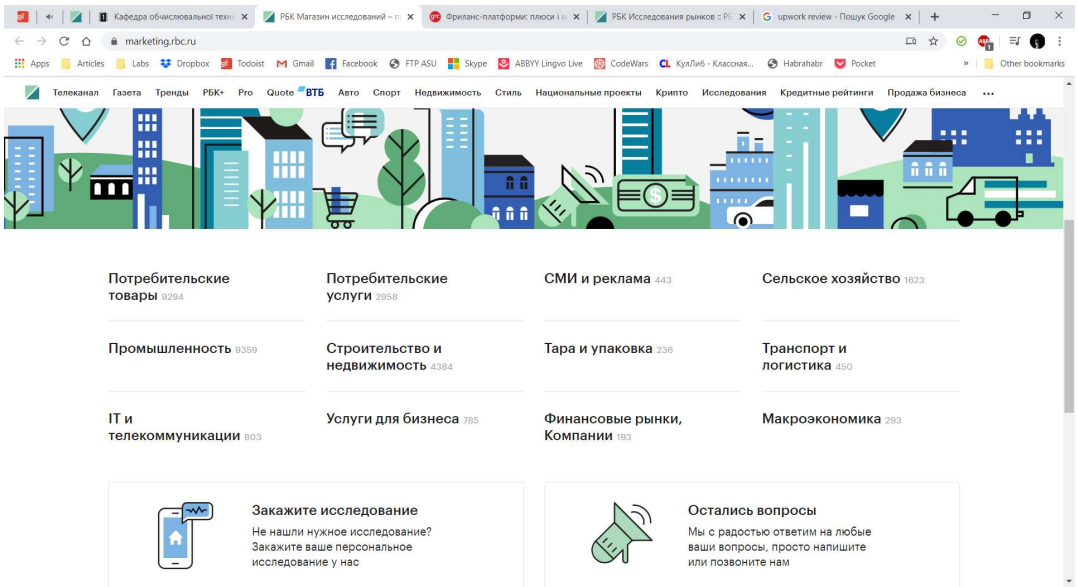


Рис. 1.6. Головна сторінка сайту marketing.rbc.ru

На рис. 1.6. можна побачити основні категорії доступні для здійснення досліджень. Реєстрація підтримує можливість створення корпоративного аккаунту компанії і персонального аккаунту користувача. Ресурс має

можливість оплати завдання та взаємодії з клієнтами через повідомлення. В ході огляду вдалось визначити наступні мінуси:

- Можливість здійснення виключно маркетингових досліджень
- Відсутня можливість розділення маркетингового дослідження на частини
- Даний сайт працює лише в Російській Федерації

1.4. Порівняльна таблиця існуючих засобів

У табл. 1.1 описано функціонал та його підтримку у описаних вище існуючих засобах.

Таблиця 1.1

Порівняння існуючих засобів

Функціонал	Freelancer .com	Upwork	Marketing. rbc.ru	Розроблений продукт
Створення дослідження	+	+	+	+
Декомпозиція дослідження на частини	+	+	-	+
Оплата дослідження за виконану частину	-	+	-	+
Захист виконавця від випадку, в якому його робота залишиться неоплаченою (у разі успішного виконання завдання)	-	-	-	+

Можливість обговорення пропозиції по завданням	+	+	+	+
Захищене збереження результатів виконання досліджень за допомогою алгоритму Blockchain	-	-	-	+

У результаті проведеного дослідження, ми можемо побачити, що розроблений продукт має переваги над існуючими аналогами у наступних пунктах:

- Оплата дослідження за виконану частину
- Захист виконавця від випадку, в якому його робота залишиться неоплаченою
- Захищене збереження результатів виконання досліджень за допомогою алгоритму Blockchain.

ВИСНОВОК ДО РОЗДІЛУ 1.

У першому розділі магістерської дисертації було оглянуто основні продукти, які є аналогами до розробленого рішення:

- Freelancer.com
- Upwork
- Marketing.rbc.ru

У ході знайомства та огляду вищевказаних програмних продуктів, які реалізовують схожий функціонал, було виявлено важливі недоліки цих систем:

- Немає можливості розділення завдання на частини
- Відсутність оплати за виконану частину завдання
- Відсутність регуляції виконання завдання поетапно
- Виконавець має шанс залишитись без оплати за виконану роботу.
- Немає змоги переглядати тимчасові результати роботи.

Таким чином, актуальною є задача розробки власної системи для продажу послуг та досліджень.

РОЗДІЛ 2.

ВИБІР АРХІТЕКТУРИ ТА ІНСТРУМЕНТІВ

Важко переоцінити важливість вибору правильної архітектури, основних інструментів та підходів до проектування для розробки додатку. Дані рішення відіграють велике значення у майбутньому проекту, впливаючи на складність внесення змін у майбутньому, ціні імплементації та підтримку. Провідні наукові дослідження вказують на те, що вартість помилок в архітектурі є досить високою. Дорожче обходяться тільки помилки у вимогах. Саме з цієї причини вдалі архітектурні рішення вигідні, як для бізнесу, так як коштують дешевше, так і для розробників, тому що з ними легше працювати.

Архітектура – це не тільки зліпок того, що вже є або візія процесу розробки програмного забезпечення за конкретними вимогами, це погляд у майбутнє, пристосування обчислюваної системи до викликів, які стануть перед нею на фазі експлуатації.

В ході проектування предмету розробки магістерської дисертації було обрано реалізувати його як клієнт-серверний веб-додаток [6]. З даного вибору випливає те, що в якості сервера в нас буде виступати IIS (традиційний вибір для платформи .NET), а клієнтом буде веб-браузер, в якому користувач зможе відкрити клієнтський веб-сайт. Взаємодія між цими компонентами проходить переважно через протокол HTTP та веб-сокети.

Клієнтський веб-сайт надає можливість користувачу взаємодіяти з веб-сервером через зручну, графічну обгортку, яка у результаті взаємодії продукує запити до веб-серверу. Також в деяких випадках клієнтський веб-сайт здійснює просту перевірку даних, наприклад, на те, чи користувач ввів якісь символи в поле.

Веб-сервер, у даній моделі, обробляє HTTP-запити надіслані клієнтом та реалізує певні логічні дії над даними, які зберігаються в БД, записуючи оновлені дані при потребі, та повертаючи результати обробки в тілі відповіді на HTTP-запит.

2.1. Особливості архітектури додатку

На стадії проектування магістерської роботи було прийнято реалізувати додаток, використовуючи трьохрівневу архітектуру. Це частковий випадок клієнт-серверної архітектури. Оглянемо детальніше складові частини даного архітектурного підходу на конкретному прикладі.

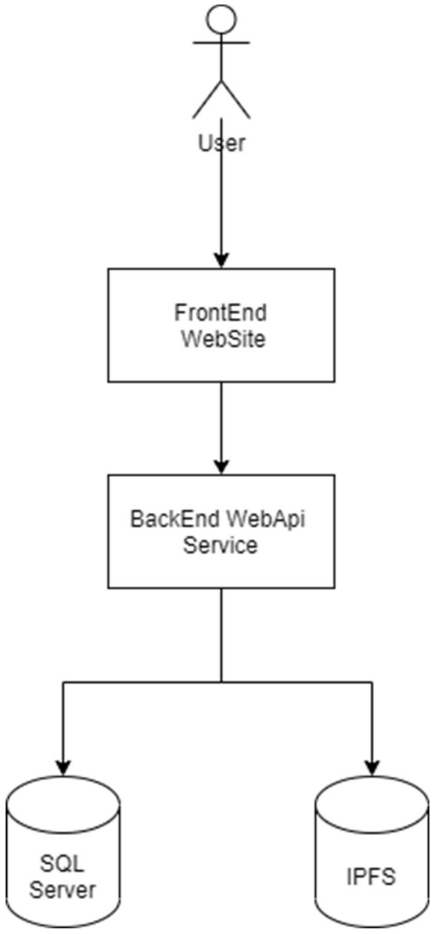


Рис. 2.1. Поточна архітектура додатку

На рис. 2.1 зображено поточну архітектуру додатку. Як ми бачимо по архітектурній схемі у ній є наступні складові:

- Клієнтський рівень – основним завданням даного рівню є надання зручного користувацького інтерфейсу для взаємодії з програмою. За допомогою функціоналу реалізованому в цьому рівні користувач не потребує прямої роботи з серверним рівнем, а має зручну і гнучку

надбудову над ним. Вона займається обробкою подій, які створює користувач – натискання на кнопки, перехід між сторінками, базовий ввід клавіш з клавіатури, а також надсилає запити на сервер, інформуючи його про виконані дії. У нашому випадку, для розробки клієнтського рівню було використано фреймворк Angular, а також HTML, CSS, JavaScript та TypeScript.

- Серверний рівень – реалізує основні бізнес-правила роботи додатку, обробляє запити надіслані з клієнтської частини додатку, робить певні перетворення на основі отриманих даних та записує результати виконання запитів в певну БД. Відіграє провідне місце в даному архітектурному підході, так як зв’язує між собою два важливих компонента – БД і клієнтську частину. Для реалізації предмету розробку магістерської роботи було обрано платформу .NET Core.
- База даних – відповідає за збереження певного стану даних в додатку. На даний момент в розробці програмного забезпечення БД діляться на дві великих категорії: SQL і NoSQL. Для розробки поточного рішення було обрано базу даних MS SQL Server.

В порівнянні з іншими архітектурними підходами або їх відсутністю трьохрівнева архітектура має суттєві переваги:

- Зменшення часу розробки продукту - можна зручно розділити розробку додатку між різними командами, наприклад, віддати серверну частину - одній команді, клієнтську частину – іншій, попередньо узгодивши контракти між різними рівнями.
- Чітке розподілення відповідальності – при правильно спроектованому рішенні буде легше знаходити дефекти, локалізувати проблеми з продуктивністю, оптимізувати додаток.
- Розширюваність – при необхідності, можна збільшити кількість додатків серверного рівня, щоб краще справлятись з навантаженням.

2.1.1. Вибір архітектури клієнтської частини додатку

Оглянемо інструменти та архітектурні підходи, які були використані для розробки клієнтської частини додатку. Вони слугують для виконання основних завдань додатку – надання зручного і швидкого користувацького інтерфейсу, з'єднання та обміну даними з сервером за допомогою протоколу HTTP, обробки помилок, які можуть виникнути у самому клієнтському додатку, так і при взаємодії з серверним рівнем.

Для ефективної проектування та розробки клієнтського веб-сайту було обрано односторінковий підхід (SPA) [7]. Суть цього підходу полягає в тому, що у нас є тільки одна сторінка, яка завантажується у веб-браузер. Далі при виконанні різноманітних дій, наприклад, переходів, ми не беремо нову сторінку з сервера, а проводимо переписування поточної сторінки. Попри певні недоліки даного підходу (завантаження першої сторінки може зайняти трохи більше часу), є суттєві переваги:

- Чітко відділяється відповідальність між клієнтом та сервером
- Робота додатку пришвидшується за рахунок того, що при завантаженні першої сторінки, ми отримуємо більшість залежностей, які потім кешуються в браузері. Надалі клієнтський додаток взаємодіє з сервером тільки в плані виконання HTTP-запитів з необхідними даними для виконання певних операцій або для того, щоб взяти лише дані, а не повну сторінку.

Так як, реалізація функціоналу, який надають більшість наявних сьогодні бібліотек є досить довгою і копіткою працею, зазвичай, використовують уже готові фреймворки для розробки на їх базі SPA-додатків. До найбільш відомих та використовуваних SPA-фрейморків, на даний момент, відносять:

- React
- Angular
- VueJS

Для створення клієнтської частини додатку було прийнято рішення використати фреймворк Angular [8]. Angular – це платформа для розробки ефективних та складних односторінкових додатків засобами HTML, CSS, JavaScript та TypeScript.

Основним будівельним блоком Angular можна назвати таку сутність, як компонент. Компонент являє собою будівельний блок, який містить у собі розмітку сторінки, стилі та власне код логіки роботи компоненту. Компонент може бути перевикористаний, наприклад, в іншому компоненті. Також варто відмітити, що компоненти можуть взаємодіяти один з одним через події. Навігація в додатку забезпечується гнучким механізмом маршрутизації. Ми можемо прикріпити до певного маршруту відповідний компонент, а такою передати в нього різноманітні параметри.

Для винесення логіки, яка може бути використана в різних компонентах або в інших складових додатку використовують сервіси. У нашому випадку, в окремі сервіси була винесена логіка роботи з даними користувача та генератор унікальних ідентифікаторів. Компоненти можуть отримати необхідні сервіси за допомогою механізму ін’єкції залежностей [9].

У порівнянні з іншими аналогами було виділено ряд суттєвих переваг:

- Компоненто-орієнтованість
- Більшість паттернів вбудованих в фреймворк дозволяють спростити роботу та зменшити ймовірність помилок допущених у проектуванні
- Зручні механізми для роботи з DOM
- Популярність
- Вбудована підтримка мови TypeScript

Загалом, в написанні логіки клієнтського рівня додатку широко використовувалась мова програмування TypeScript, створена в 2012 році Андерсом Гейлсбергом. Тоді було зрозуміло, що у JavaScript є певний ряд недоліків, наприклад, складність підтримки клієнтських додатків при великій кількості коду та незрозумілих контрактах між частинами додатку [10]. Саме тоді розробникам прийшла на думку ідея створити мову програмування, яка б

могла бути легко зворотньо сумісною з JavaScript, але мала статичну типізацію. Такі нововведення допомогли підвищити швидкість розробки, зменшити час на відловлювання помилок та рефакторінг, збільшили можливість перевикористання коду. Так як, TypeScript компілюється в JavaScript, то він може виконуватись на всіх тих платформах, на яких може виконуватись JavaScript. З часом, такі концепції як модульність та класи були реалізовані і в мові JavaScript.

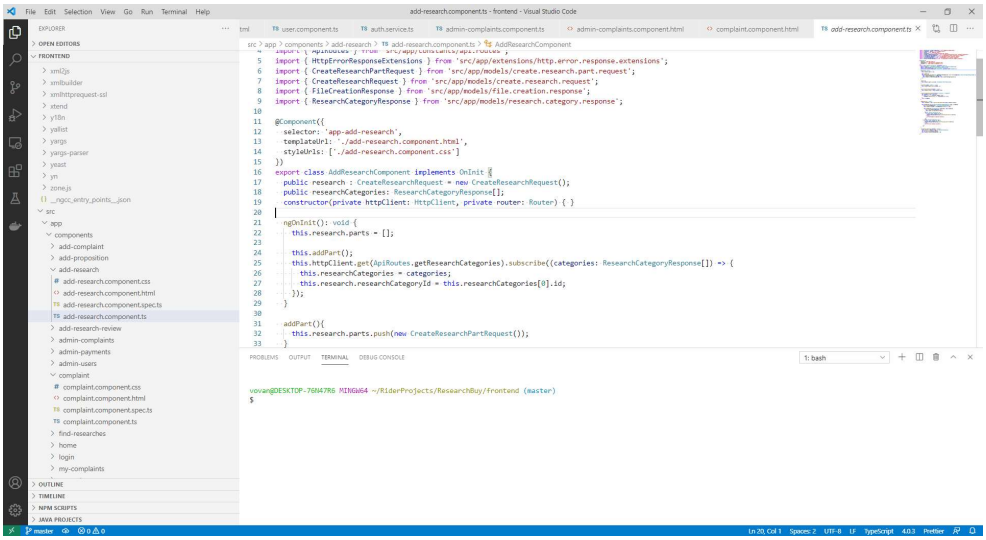


Рис. 2.2. Головне вікно редактора Visual Studio Code

Для розробки клієнтської частини додатку було використано Visual Studio Code (рис. 2.2). Це сучасний текстовий редактор, який використовують для створення веб-застосунків [11]. Він має велику спільноту та багато плагінів, які дозволяють суттєво покращити цей інструмент та надати йому нові властивості. Visual Studio Code підтримує більшість сучасних мов: JavaScript, C#, TypeScript, Java, Kotlin, Python, C++, а також інші. Варто відмітити, що цей додаток є може бути встановлений, як на Windows, так і на MacOS або Linux.

Варто відмітити, що в даний продукт інтегрований термінал, тому для виконання команд не потрібне окреме вікно. Для роботи з Angular це в якійсь мірі критично, так як для роботи з клієнтським додатком активно використовується angular-cli – набір консольних команд для роботи з додатками на базі фреймворку Angular.

2.1.2. Вибір архітектури серверної частини додатку

Розглянемо архітектурні рішення та інструменти на серверній частині додатку. Як було сказано вище, даний рівень призначений для того, щоб ефективно приймати запити від клієнтської частини додатку, обробляти їх згідно певних бізнес-правил та на основі цієї логіки проводити операції по збереженню та опрацюванню інформації з БД. У нашому випадку, серверний додаток – це програма, яка написана на платформі .NET Core, з використанням сучасної бібліотеки для розробки веб-додатків ASP.NET Core на мові програмування C#.

Історично, платформа .NET розроблялась компанією Microsoft. Вона була досить закритою в плані імплементації. З кожним оновленням випускались нові версії .NET Framework, який працював лише на ОС Windows. Проте, приблизно в 2014 році, корпорація зрозуміла, що від того, що код закритий платформа тільки програє. З того часу почався рух на відкриття платформи. У 2016 році відбувся перший реліз .NET Core [12] і це в певній мірі було проривом і значним поштовхом до позитивних змін. Код .NET Core став відкритим і відповідно спеціалісти з усього світу, які користуються .NET змогли вносити зміни у місця, які можна було б покращити. Це привело до чудових результатів: .NET Core тепер можна використовувати як на Windows, так і на MacOS, і на Linux, значно виросла продуктивність та зручність написання та підтримки коду. Саме тому, для розробки серверної частини додатку було обрано .NET Core.

.NET Core складається з наступних складових:

- CoreCLR – вдосконалена та перероблена імплементація CLR, яка була створена Microsoft, як середовище для управління виконанням програм на платформі .NET Core.
- RyuJIT – дороблена та покращена версія JIT-компілятора з .NET Framework.

- CoreFX – часткова копія стандартних бібліотек .NET Framework, яка також була вдосконалена для .NET Core.

Також варто відзначити, що при установці .NET Core автоматично стає доступним набір консольних команд для розгортання, збірки та запуску тестів. При необхідності, можна завантажити додаткові плагіни, які розширять фунціонал стандартної команди dotnet.

Таблиця 2.1

Порівняльна характеристика .NET Core та .NET Framework.

Властивості	**.NET Core**	**.NET Framework**
Підтримувані ОС	Windows, Linux, Mac OS	Windows
Доступ до коду платформи	Відкритий на GitHub	З початку розробки був закритим, але згодом почав відкриватись
Мови	C#, F#	C#, F#, Visual Basic
Продуктивність	В порівнянні, з .NET Framework продуктивність платформи значно виросла	В порівнянні, з .NET Core продуктивність значно нижча.
Підтримка Docker	+	+
Зручність розробки	+	-
Рекомендації від Microsoft	Загалом, для нових додатків, яким необхідна кросс-платформовість, Docker, мікросервіси, висока продуктивність та розширюваність системи.	Використовувати коли є існуючий додаток на .NET Framework або міграція на .NET Core є досить складною через функціонал, який не підтримується в ньому або через пакети, які зараз недоступні на .NET Core

В результаті порівняння даних платформ (табл. 2.1), ми визначили, що системи на .NET Core є більш продуктивними з точки зору платформи. Про це також свідчать рекомендації від Microsoft.

На базі .NET Core можна створювати різні типи додатків: консольні, веб-додатки, UWP. У нашому випадку, було прийнято рішення створити веб-додаток на серверній частині з використанням бібліотеки ASP.NET Core [13], так як вона надає швидкий, безпечний та зручний інтерфейс для обробки HTTP-запитів.

Однією з основних переваг ASP.NET Core є її модульність. При потребі, ми можемо підключати необхідні нам складові, не обтяжуючи програму непотрібними залежностями. Такої можливості, в ASP.NET, на жаль не було. В ASP.NET Core є можливість створення WebAPI та MVC додатків. Так як, клієнтська частина рішення повністю відокремлена від серверної, то було вирішено розробити Web API з використанням підходу REST.

REST – це архітектурний підхід до побудови серверних веб-додатків. Він був розроблений Роєм Філдінгом ще в 2000 році [14]. Базується даний підхід на наступних принципах:

- Взаємодії проходять у клієнт-серверній моделі.
- Веб-фреймворк не зберігає стану, на відміну від інших підходів, в яких стан може зберігатись в серверних сесіях або cookies. Дані передаються виключно з запитом надісланим з клієнта.
- Можливість налаштування ефективних механізмів кешування.
- REST-система надає уніфікований та стандартизований інтерфейс для взаємодії з системою. Наприклад, при запиті з HTTP-методом GET ми очікувано отримаємо певні дані, запит з методом POST здійснить запис певних даних або виконання дії, PUT – модифікує існуючу модель даних, DELETE – видалить дані про певний ресурс на сервері.
- Додаток є розділеним на певну кількість компонентів. Самостійні компоненти не можуть взаємодіяти з елементами, які є поза зоною їхньої видимості.

Центральною сутністю в REST-підході є ресурс. Його зазвичай можна ідентифікувати за допомогою URL, який побудований відповідно до стандартів REST. Також варто відмітити те, що у традиційних REST-системах ресурс може мати відображення у різних форматах найпопулярнішими з яких є JSON та XML. Зазвичай, REST-додатки підтримують значно більшу кількість форматів.

Для імплементації серверної частини програми було використано мову програмування C# - це строго типізована об'єктно-орієнтовна мова програмування загального призначення [15]. Основною метою розробки даної мови було підвищення продуктивності програмістів. Головним ідейним натхненником та архітектором з першої версії є Андерс Гейлсберг, який також прийняв участь у розробці Turbo Pascal, Delphi та TypeScript. C# не є залежним від платформи, його код може бути виконаний на таких ОС як Windows, Linux, MacOS та інші. Звичайно, як і більшість об'єктно-орієнтовних мов, C# підтримує механізм інкапсуляції, наслідування та поліморфізму, проте в нього є певні властивості, які вирізняють C# серед інших об'єктно-орієнтованих мов [16]:

- В C# кожен тип унаслідується від object. Це означає, що кожен з типів має певний спільний базовий функціонал.
- Впровадження інтерфейсів.
- Синтаксичний цукор – властивості та події.
- Методи можуть передаватись як значення через механізм делегатів.
- Підтримка шаблонів для зменшення ризиків побічних ефектів у структурах, які вважаються розробником незмінними.

В сучасних реаліях, C# є однією з найпопулярніших мов, не тільки через зручність написання коду, а й через величезне різноманіття типів додатків, які можна на ньому розробляти:

- Консольні додатки
- Додатки для робочих станцій (Windows Forms, WPF, UWP)
- Додатки для мобільних пристроїв (Xamarin)
- Розробка кросс-платформових ігор (Unity)

- Веб-додатки та веб-сервіси (ASP.NET, ASP.NET Core, WCF)

Після визначення основних архітектурних концепцій для шару додатку, перед розробником постало питання вибору інструменту для розробки продукту. Було обрано інтегроване середовище для розробки програмного забезпечення JetBrains Rider. Першу версію було анонсовано в 2016 році.

Даний інструмент надає зручні засоби для розробки додатків на .NET Framework та .NET Core, можливість створювати консольні, десктопні, мобільні та веб-додатки. Основною перевагою перед головним конкурентом, Visual Studio 2017 у зв'язці з плагіном JetBrains Resharper, є його доступність не тільки на Windows, а й на Linux та MacOS.

Інструмент надає широкі можливості для навігації, виправлення та переробки коду. Однією з головних переваг, є наявність вбудованого терміналу та системи контролю версій. Також у даний продукт вбудовано профайлер для відслідкування проблем з продуктивністю розроблюваного додатку. В порівнянні з Visual Studio, установка та управління пакетами, працює швидше і краще. Варто відзначити, що у Rider вбудований засіб для декомпіляції коду DotPeek, що значно дозволяє скоротити час у разі відладки бібліотечного коду.

2.1.3. Вибір архітектури рівню бази даних

При розробці додатку одним з першочергових і важливих питань, яке постало перед розробниками, був вибір бази даних. База даних – це набір упорядкованих, логічно взаємопов'язаних даних, які призначені для задоволення інформаційних потреб користувачів. На сьогоднішній день, є дві великі категорії баз даних на які ділиться сучасна індустрія розробки програмного забезпечення: SQL (реляційні) та NoSQL (нереляційні) [17]. Існує велике різноманіття нереляційних даних. Зазвичай, виділяють наступні категорії:

- БД для збереження даних у форматі ключ-значення – хороший варіант для кешуваннння даних. Приклад, Redis.

- Графові – бази даних, призначені для збереження інформації у вигляді графів. Приклад, Neo4J.
- Документо-орієнтовані – бази даних, призначені для збереження інформації у ієрархічному вигляді та, зазвичай, денормалізованому вигляді. Приклад, MongoDB.
- Колонко-орієнтовані – бази даних, які зберігають інформацію радше за колонками, ніж за стовпчиками. Приклад, Cassandra.

Ринок доступних рішень, які надають реляційні бази даних є досить великим і сформувався досить давно. Реляційна модель баз даних сформувалась на базі 12 правил Едварда Кодда, які він сформулював в 1970 році [18]. На жаль, всі 12 правил до цих пір не забезпечує жодна БД. Сьогодні, БД визначають як реляційну, якщо БД:

- Представляє дані у вигляді набору таблиць, які складаються з певної послідовності рядків та стовпчиків, з унікальним ключем.
- Дає можливість оперувати даним за допомогою реляційних операторів.

В порівнянні, з NoSQL базами даних, у реляційних баз даних є можливість зручного налаштування зв’язків між таблицями та введення певних обмежень (constraint) на значення в полях. Зазвичай, доступні наступні типи обмежень:

- NOT NULL – обмеження, яке не дозволяє записувати значення NULL у значення певного стовпчика в рядках таблиці
- UNIQUE – обмеження, яке забезпечує унікальність значень в межах однієї колонки.
- PRIMARY KEY – обмеження, яке, по суті, являється комбінацією NOT NULL та UNIQUE. Його суть полягає в тому, що за значенням поля, яке помічене цим обмеженням, можна визначити, до якого рядку воно належить. Зазвичай, при створенні первинного ключа автоматично створюється первинний індекс.

- FOREIGN KEY – обмеження, яка спрямоване на перевірку значень в таблиці, які посилаються на дані іншої таблиці, таким чином, щоб у початковій таблиці не було даних, яких немає в таблиці на яку ми посилаємось.
- CHECK – обмеження, яке здійснює перевірку на виконання певної умови.
- DEFAULT - обмеження, яке, за замовчуванням, присвоює комірці певне значення.
- INDEX – обмеження, яке дозволяє оптимізувати швидкість отримання даних. В реляційних БД існує 2 типи індексів: кластеризовані та некластеризані. Основна відмінність в тому, що кластеризований індекс може бути тільки один на таблицю, при цьому він визначає фізичну структуру розташування даних в БД. Для побудови некластеризованих індексів, зазвичай, використовуються такі структури як B-tree, та їх вдосконалені аналоги.

Інколи для оптимізації швидкодії використовують такі сутності реляційних БД, як збережені процедури. Збережені процедури – це код на мові програмування SQL, який зберігається на стороні БД. Він може бути виконаний і перевикористаний велику кількість разів. За рахунок того, що код зберігається на стороні БД, він уже максимально оптимізований до виконання.

Також варто відмітити, що реляційні БД надають можливість проведення транзакцій. Транзакція – це послідовність дій, які можуть бути виконаними над певним об’єктом або об’єктами, які сприймаються як одне ціле. Тобто, якщо з якихось причин код, огорнутий в транзакційну оболонку не може бути виконаним, то всі попередні зміни будуть відмінені. Це має як очевидну перевагу – у разі помилки дані не будуть пошкоджені, так і один недостаток – коли виконується транзакція, здійснюється блокування таблиці, яке може спричинити зниження швидкодії виконання запитів. Загалом, транзакціям притаманні наступні властивості (ACID) [19]:

- Атомарність

- Консистентність
- Ізольованість
- Довговічність

В наш час, SQL став фактично стандартною мовою, на якій працюють більшість реляційних БД. Основним призначенням даної мови є надання зручного та ефективного інструменту для написання запитів та роботи з БД. Традиційно, SQL можна розділити на наступні підмови:

- DDL (Data Definition Language) – мова, які використовується для визначення структури даних. З її допомогою ми маємо можливість створювати таблиці, схеми, процедури. Ключові слова: CREATE, DROP, ALTER.
- DQL (Data Query Language) – мова, що використовується для написання запитів до БД. Ключові слова: SELECT, FROM, WHERE, LIKE, JOIN.
- DML (Data Manipulation Language) – мова, для здійснення змін в існуючі дані. Ключові слова: INSERT, UPDATE, DELETE.
- DCL (Data Control Language) – мова, яка надає можливість для контролю доступу до даних. Ключові слова: GRANT, REVOKE.

Загалом, всі основні гравці на ринку реляційних БД підтримують стандарт SQL. Хоча, майже кожен з них надає розширення до стандартної мови SQL. Найбільш відомими розширеннями є T-SQL, SQL PL, PL/SQL.

Через окреслені вище властивості та переваги: надання зручного та зрозумілого інтерфейс для роботи з структурованими даними, підтримка ACID-транзакцій, розширена робота зі зв’язками, налаштування обмежень, ефективні механізми індексації – було прийнято рішення використати SQL Server, як основну БД для роботи з бізнес-даними.

SQL Server підтримує позитивні сторони описані вище. Сьогодні SQL Server можна встановити не тільки на ОС Windows, а і на Linux, та Mac OS. Існує можливість розгортання SQL Server у середовищі для роботи з контейнеризованими додатками – Docker. Також SQL Server реалізує діалект

мови SQL – T-SQL, який надає розширені можливості для роботи з БД. Для роботи з SQL Server та іншими реляційними БД було розроблено бібліотеку Entity Framework Core. Це облегшена та покращена версія Entity Framework, яка ідеально підходить для роботи з SQL Server за рахунок широкої підтримки його функціоналу.

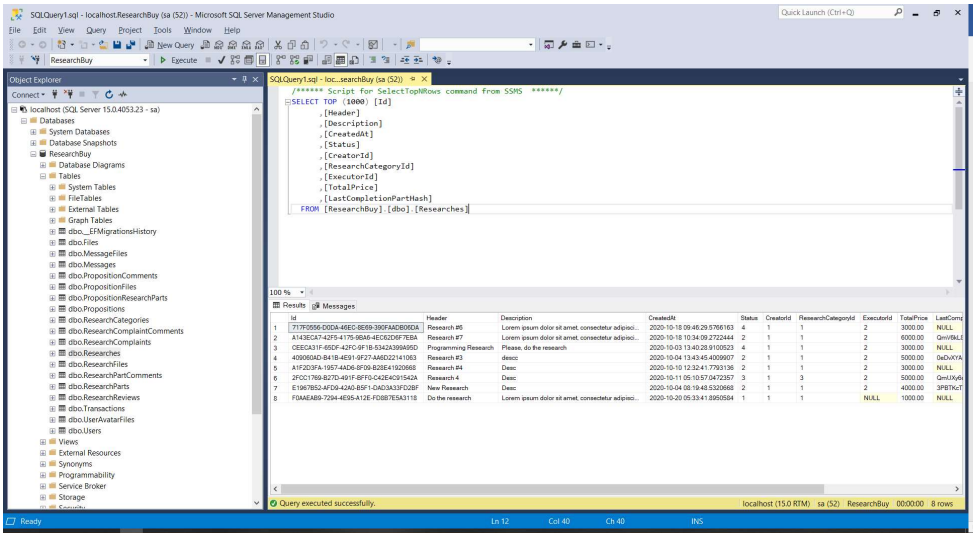


Рис. 2.3. Вікно середовища для роботи з SQL Server - SQL Server Management Studio

На рис. 2.3 зображено вікно програмного продукту для роботи з SQL Server – SQL Server Management Studio. Даний інструмент надає чудові можливості для роботи з БД. Є можливість з’єднання з сервером БД та перегляду його вмісту. При перегляді БД реалізований функціонал зручний графічний інтерфейс для виконання запитів, створення таблиць та індексів, налаштування можливостей доступу до БД.

ВИСНОВОК ДО РОЗДІЛУ 2

Отже, після обміркування та визначення основних властивостей системи: висока продуктивність, захищеність, доступність та легкість експлуатації було прийняте рішення реалізувати веб-додаток із застосуванням трирівневого архітектурного підходу та використати для кожного з шарів наступні технології:

- Рівень клієнтського додатку – односторінковий додаток, написаний з використанням веб-фреймворку Angular, на мові програмування TypeScript, в інтегрованому середовищі Visual Studio Code.
- Рівень серверного додатку – додаток, на ASP.NET Core з використанням підходу REST API, на мові програмування C#, в інтегрованому середовищі JetBrains Rider.
- Рівень бази даних – реляційна БД MS SQL Server.

РОЗДІЛ 3.

ОПИС МЕТОДІВ ТА АЛГОРИТМІВ ЗАХИСТУ ДАНИХ В ДОДАТКУ

В сучасних додатках захист даних та інформації є невід'ємною частиною успішного проекту. Сьогодні у розробці програмного забезпечення поширена тенденція до забезпечення якомога більшого рівню захисту: частіше стали з'являтись додатки, які досить наполегливо пропонують подвійну аутентифікацію, більшість публічних сайтів в мережі інтернет використовують захищені протоки, наприклад HTTPS, для створення паролів існують спеціальні правила, щоб зробити їх підбір якомога слабшим. Ці всі міри – виправдані. Адже, якщо середній користувач лише інколи задумується про безпеку своїх даних, то розробники мають це зробити за нього. Перейдемо до опису методів та алгоритмів захисту даних реалізованих в поточному додатку.

3.1. Аутентифікація та авторизація дій користувача

Одним з перших питань, яке виникло на початку проектування роботи додатку було: "Як визначити хто зараз перед мною?". Наступним питанням було: «Чи може поточний користувач здійснити цю дію?». Перше питання вирішують механізми аутентифікації, а друге – авторизації.

В даній програмі використовується аутентифікація та авторизація користувача через JWT-токен. JWT (JSON Web Token) – це відкритий стандарт (RFC 7519), який визначає компактний спосіб для захищеної передачі даних між застосунками у вигляді JSON об'єкту [20]. Інформація в токені може бути перевірена, тому що в ній використовуються засоби цифрового підпису. Він складається з 3 частин:

- Заголовок – об'єкт, що містить інформацію про тип токен і алгоритм його шифрування.

- Тіло – об’єкт, що містить дані, необхідні для авторизації користувача. Також може містити певну додаткову інформацію, наприклад, ідентифікатор користувача, адресу електронної пошти, ім’я.
- Підпис – рядок, що створюються за допомогою секретного коду, заголовку та тіла. Зазвичай, це поле використовують для перевірки токена.

В порівнянні з таким способами аутентифікації, як SAML (Security Assertion Markup Language Tokens) і SWT (Simple Web Tokens), JWT має наступні переваги:

- Компактність – формат JWT базується на форматі JSON, який набагато більш компактний, ніж XML-формат, який використовується в SAML і SWT.
- Зручність – більшість мов мають досить потужні вбудовані парсери, для JSON, які працюють ефективніше в порівнянні з XML.
- Гнучкість – в порівнняні з SWT, який може бути підписаний виключно симетричним ключом, використовуючи певний спільний секрет, JWT I SAML можуть використовувати для підпису алгоритми RSA та ECDSA, які в свою чергу зможуть взяти пару публічний/приватний ключ з X.509 сертифікату.

У поточній реалізації, JWT-токен створюється при вході користувача в систему. Якщо пошта користувача та хеш-паролю співпадають, то відповідно починаємо створювати JWT-токен. При створенні токену, ми маємо можливість надати додаткові дані, які ми зможемо використовувати при аутентифікації користувача на подальших запитах. Відповідно, цю нагоду ми використовуємо і передаємо туди:

- Ідентифікатор користувача
- Електронну пошту
- Повне ім’я
- Роль

Після надання додаткових даних, які у термінології JWT будуть називатись claims, ми можемо налаштувати ще кілька полів. Можна вказати час після надання якого токен стане недійсним. В нашій реалізації це буде 7 днів від створення токену. Перед створенням токену також вкажемо алгоритму для підпису – HMACSHA256. Після проведення операції токен передається на клієнтську частину додатку. Відповідно, всі наступні запити на сервер будуть приходити зі спеціальних HTTP-заголовком Authorization, в якому буде міститись поточний JWT-токен, який зараз є на клієнті.

Для того, щоб визначати особистість користувача при кожному запиті потрібно підключити бібліотеку Microsoft.AspNetCore.Authentication.JwtBearer [21]. У поточній імплементації ASP.NET Core існує таке поняття як middleware. Суть цього поняття полягає в тому, що даний механізм дозволяє зробити певні дії з кожним HTTP-запитом, який надійшов в програму. З них, зазвичай, будуються цілі ланцюжки, які дозволяють забезпечувати логування запитів, обробку певних виключень та багато чого іншого. Для підключення механізму аутентифікації, потрібно викликати метод UseAuthentication, таким чином ввімкнувши middleware, який відповідає за аутентифікацію.

Для забезпечення механізму авторизації користувача в ASP.NET Core використовують атрибут [Authorize] та метод для підключення middleware UseAuthorization, який знаходиться в бібліотеці Microsoft.AspNetCore.Authorization. Після позначення методу в контролері цим атрибутом неавторизований користувач не зможе провести операцію або взяти дані, які для нього недозволені.

3.2. Збереження результатів досліджень у захищеній файловій системі IPFS з використанням Blockchain

При плануванні розробки дипломної роботи, перед нами постало складне завдання – захист результатів виконання дослідження. Це питання є досить

важливим, так як витік даної інформації у недобросовісні руки може обернутись бідою, як для замовників, так і для виконавців.

При виборі рішення для збереження результатів виконання завдання було обрано IPFS. IPFS – це peer-to-peer система для обміну файлами, яка фундаментально відрізняється у тому, як інформація розподіляється між частинами системи [22]. IPFS містить певні інновації у підході до побудов комунікаційних протоколів та розподілених систем.

Розробка IPFS почалась зі спроби Джуана Бенета, ціллю якої було створення системи, яка б дозволяла швидко переправляти версіоновані наукові дані. Для розробки було використано синтез таких технологій як DHT, системи контролю версій Git та Bittorrent.

У основі IPFS лежать 3 основних фундаментальних принципи:

- Ідентифікація унікальності через адресування вмісту
- Зв’язка вмісту через направлені ациклічні графи (DAGs)
- Знаходження контенту через розподілені хеш-таблиці (DHTs)

Адресування вмісту використовується в IPFS для того, щоб ідентифікувати контент більше по тому, що в ньому знаходиться, ніж по тому, де він знаходиться. Пошук контенту по вміст досить типова задача для сьогодення. Наприклад, здійснюючи покупки в магазині ми радше робимо пошук по вмісту, а не по місцю, де знаходиться цей продукт. Ця проблема існує, зокрема, в сучасній мережі Інтернет. Для вирішення цієї проблеми кожен фрагмент контенту, який використовує IPFS протокол має унікальний ідентифікатор контенту (CID), або хеш вмісту.

Багато розподілених систем використовують адресування вмісту, використовуючи, хешування не тільки для того, щоб ідентифікувати контент, а і зв’язати його разом – від коммітів, які створюються при нових версіях коду до блокчейн технологій для роботи з криптовалютами.

У IPFS вбудовано ще один цікавий механізм, який називається IPLD. IPLD здійснює переклад між різними структурами даних, які пов’язані через хеші. Він також дає бібліотеки для комбінування підключених модулів для

того, щоб визначити шлях, селектор або запит поміж багатьма пов'язаними вузлами.

IPFS, як і багато інших розподілених систем, використовує таку структури даних як направлені ациклічні грами (DAG). Якщо бути більш точним, то IPFS використовує Merkle DAGs, в яких кожен вузол має свій унікальний ідентифікатор, який є хешом вмісту вузла, використовуючи певну хеш-функцію, наприклад SHA256. Це вносить певні висновки з даного визначення:

- Дана структура даних може бути побудована виключно починаючи з листків
- Кожен вузол в Merkle DAG є коренем піддерева і його підграф міститься в батьківському DAG.
- Вузли Merkle DAG є незмінними. Кожна корекція значення у них змінить його ідентифікатор, що спричинить зміни у всіх батьківських нодах DAG.

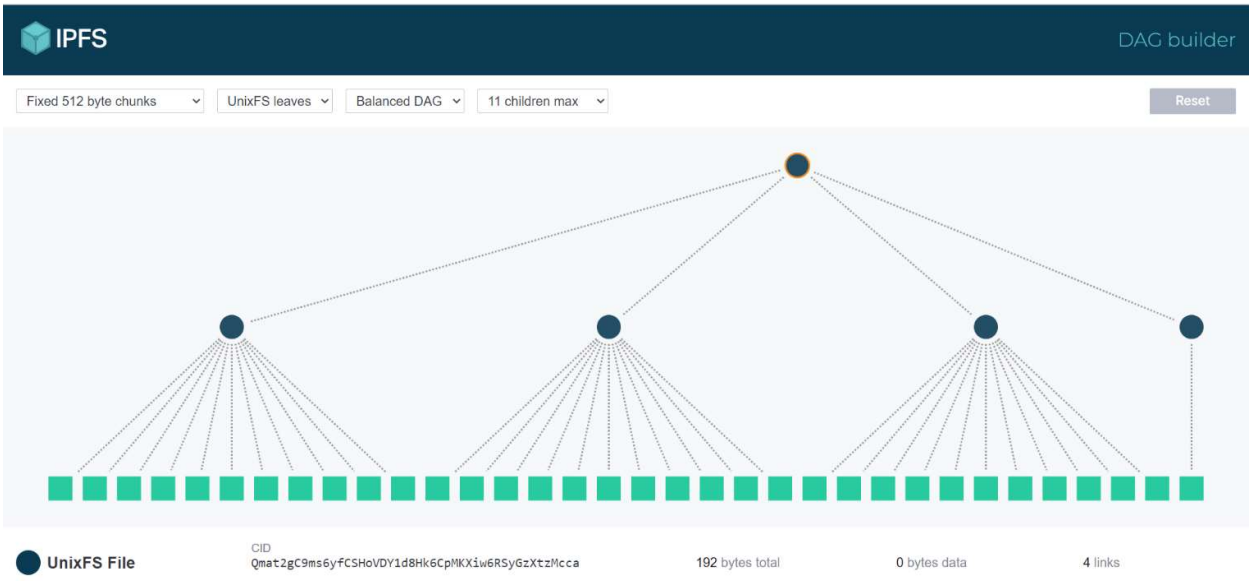


Рис. 3.1. Механізм побудови DAG на основі вхідних даних

На рис. 3.1. зображено механізм для побудови DAG на базі певного файлу. Даний інструмент є досить гнучким та дозволяє налаштувати мінімальний розмір блоку, тип DAG, кількість дочірніх елементів.

Для того, щоб визначити, які піри містять контент, який потрібно нам знайти в IPFS використовується розподілена хеш-таблиця (DHT). Хеш-таблиця, в даному випадку, є базою даних типу ключ-значення. Відповідно, розподілена хеш-таблиця розбита по всім пірам в розподіленій мережі. На рис 3.2 зображена спрощена схема роботи розподіленої хеш-таблиці.

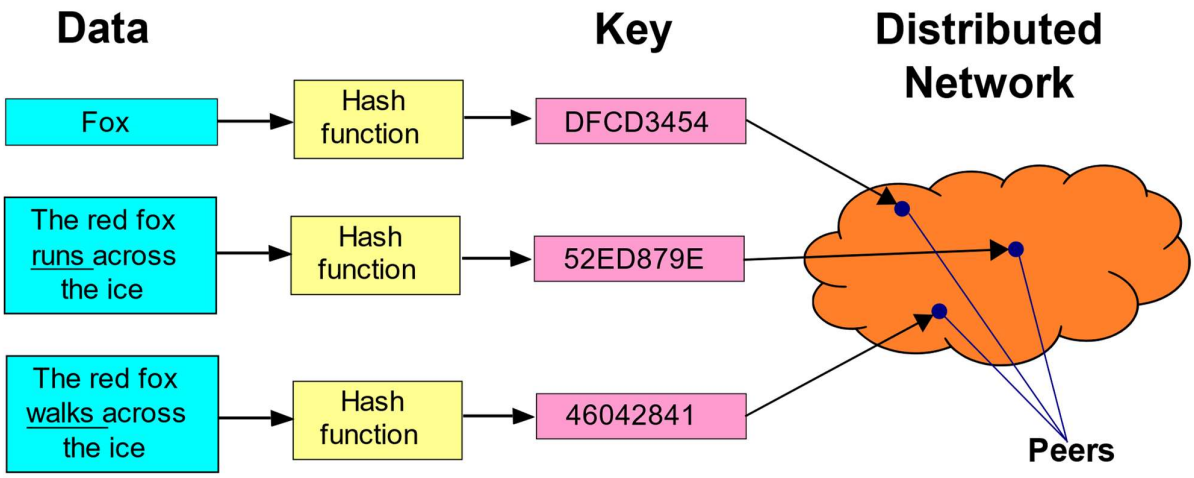


Рис. 3.2. Спрощена схема роботи розподіленої хеш-таблиці [23]

У IPFS є окремий проект – libp2p [24] – який реалізує функціонал DHT і управляє з'єднанням пірів та комунікаціями. Як тільки ми дізнаємось, де знаходиться вміст (або, точніше, які однорангові мережі зберігають кожен із блоків, що складають контент), нам потрібно ще один раз використати DHT, щоб знайти поточне місце розташування пірів. Отже, щоб перейти до вмісту, потрібно використати libp2p для запиту DHT двічі. Для того, щоб завантажити вміст, потрібно під'єднатись до контенту і викачати його. Для управління блоками інформації в IPFS використовується Bitswap, який дозволяє з'єднання з пірами, в яких міститься контент, надіслати їх у спеціальний список блоків, які нам потрібно скачати і отримати блоки, які були запитані. Після отримання блоків, ми маємо можливість перевірити їх вміст, провівши хешування вмісту і перевіривши з CIDs по яким ми запитували контент. (можна ще розписати про IPFS)

На рис. 3.3 зображений клієнтська програма для IPFS – IPFS Desktop. Дана програма дозволяє знаходити файл за CID, показує доступні піри, має можливість налаштування адреси вузла IPFS.

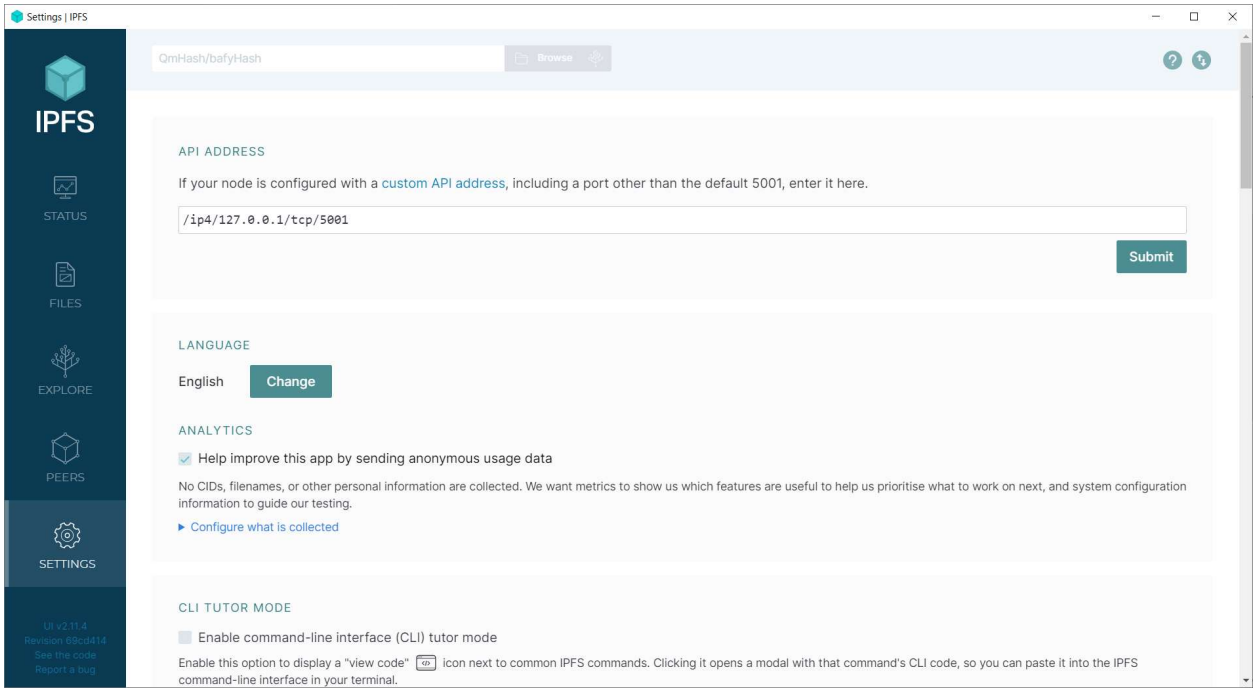


Рис. 3.3. Клієнтський додаток IPFS Desktop

Розглянемо як же зберігаються результати виконання дослідження в поточній системі. Коли виконавець завантажує на сервер результати виконання дослідження, спочатку зчитується вміст файлів.

Результати виконання дослідження зберігаються у вигляді простої реалізації Blockchain [25]. Блок містить у собі інформацію про його індекс, дату створення, хеш попереднього блоку та дані. В даних зберігаються результати виконання частини завдання: опис та файли, прикріплені, як результати виконання.

Додавання нового блоку в розробленому алгоритмі є досить тривіальним. Якщо у послідовності немає блоків, ми створюємо перший блок з індексом 0 та пустим значенням хешу попереднього блоку. Інакше, якщо у послідовності є блоки, то ми присвоюємо індексу значення на 1 більше, ніж у попередньому блоці і присвоюємо полю хешу попереднього блоку – хеш попереднього блоку.

Після заповнення даними блок серіалізується за допомогою бібліотеки Protobuf та записується в IPFS. Хеш останнього блоку записується в колонку LastCompletionPartHash в таблиці Researches. За обрахування хешу відповідає IPFS. Дана система рахує його на основі даних, які ми передаємо на запис.

По ходу розробки додатку виникла потреба забезпечити механізм зміни значення в блоках. Даний функціонал потрібен при позначенні частини завдання, як виконаної. Він модифікує значення IsAccepted в блоці з даними. У такому випадку, ми модифікуємо значення у необхідному блоці. Після цього, ми перезаписуємо даний блок в IPFS і отримуємо оновлене значення хешу для цього блоку. Відповідно, для всіх наступних блоків ми модифікуємо значення хешу попереднього блоку, записуємо його в IPFS і отримуємо хеш даного блоку. Після модифікації всіх необхідних блоків записуємо хеш останнього блоку записується в колонку LastCompletionPartHash в таблиці Researches.

При подальшій розробці проекту, за умови наявності ресурсів, планується реалізувати можливість оплати частин досліджень через криптовалюти. Відповідно, існуюча структура даних зміниться. В блок даних про дослідження можна буде додати ще хеш транзакції.

Серед великої кількості засобів для захищеного збереження результатів досліджень було обрано IPFS. Порівняємо даний інструмент з ще одним захищеним інструментом – Ethereum.

Ethereum – це криптовалюта і платформа для створення децентралізованих онлайн сервісів на базі технології блокчейн, які працюють на технології розумних контрактів. Смарт-контракт (в специфіці блокчейн) – це набір функцій і даних, які знаходять по певній адресі в блокчейні.

Основною перевагою IPFS є швидкість запису та можливість зберігати файл з великим об’ємом. Ethereum не може забезпечити настільки швидкий запис поточних блоків з результатами завдання, так як вони всередині можуть містити контент декількох файлів.

3.3. Захищене збереження конфігурацій у Hashicorp Vault

Одне з класичних завдань, яке постає перед нами, це збереження секретів додатку від стороннього доступу. Секретами, в даному випадку, можна назвати налаштування, рядки підключення до баз даних та іншу інформацію, яка може спричинити безпекові проблеми, якщо вона потрапить в руки зловмисників.

Однією з грубих помилок є збереження секретів додатку в репозиторії додатку. Для локального середовища або для середовища для розробки – це можна назвати в якійсь мірі доцільним, але збереження їх для живого середовища, на якому є персональні дані багатьох користувачів – це потенційна небезпека.

Саме для цього в багатьох додатках продумують рішення для того, щоб зберігати такі конфігурації у спеціальних захищених сховищах. Одне з таких рішень було інтегроване в дипломну роботу.

Для інтеграції було обрано продукт HashiCorp Vault. Vault – це спеціальний інструмент для збереження секретів у вигляді ключ-значення, доступ яких можна отримати у вигляді зручного і детально задокументованого REST API [26]. Секрети зберігаються у спеціальних контейнерах, в зашифрованому вигляді (AES), тому отримання самого контейнера не розкриває дані. Для різноманітних сервісів існує можливість налаштування персональних і гнучких політик доступу, щоб надавати їм тільки ті налаштування, які потрібні. Всі доступи до конфігурацій записуються в спеціальний журнал, тому при потребі, можна перевірити, хто використовував певні секрети. Також варто зазначити те, що цей продукт є безкоштовним і, на сьогоднішній день, досить широко використовується при розробці розподілених систем.

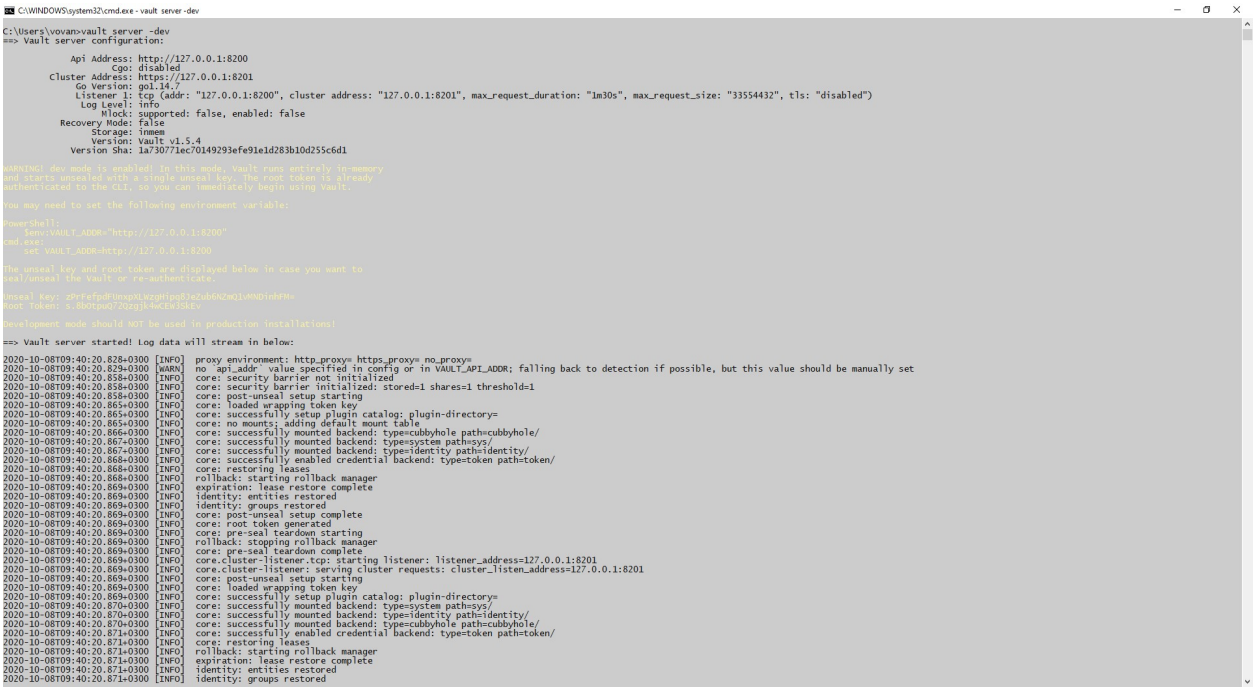


Рис. 3.4. Розгортання Vault-агента

Для початку роботи з Vault потрібно спочатку розгорнути його агент (рис 3.4). Це можна зробити, виконавши в консолі команду vault server. Зазвичай, після старту агенту генерується спеціальний токен за допомогою команди vault token create. Якщо в агенті немає даних, то ми можемо записати туди дані в форматі ключ-значення. Цю дію можна зробити, виконавши наступну інструкцію: vault kv put secret/DbSettings ConnectionString=”123”. Дістати записані значення можна, виконавши команду vault kv get secret/DbSettings.

Після заповнення агенту значеннями секретів, можна підключати наш сервіс до даного агента. Для доступу до Vault-агенту було використано бібліотеку з відкритим кодом VaultSharp. Основне її призначення – це створення обгортки – програмного інтерфейсу над REST API, який надає Hashicorp для продукту Vault. За допомогою, цієї бібліотеки легко і зручно ми можемо дістати значення з відповідного Vault-сховища, вказавши токен для аутентифікації та ключ до необхідного секрету.

3.4. Захист вихідних файлів побудови додатку від декомпіляції

Часто для того, щоб захистити вихідні файли додатку, такі як *.dll та *.exe від декомпіляції потрібні використовувати додаткові інструменти. Одним з хороших рішень на ринку є механізми, які забезпечують обфускацію коду.

Обфускація – це механізм, який гарантує ускладнення розуміння інформації, яка передається в повідомленні, коді або файлі, використовуючи нетипічні назви та символи та заплутуючи структуру повідомлення, разом з тим зберігаючи функціональність [27].

На даний момент існують обфускатори, які ускладнюють код на різних рівнях:

- Алгоритму
- Коду
- Коду ассемблеру

Розглянемо детальніше переваги даного підходу:

- Ускладнення дослідження коду та захист від експлуатації слабких місць в комерційних цілях
- Оптимізація коду

Проте, існують і певні недоліки обфускації коду:

- Втрата гнучкості коду (збільшення залежності від платформи та компілятора)
- Відладження коду стає неможливим
- Відсутність повної гарантії безпеки – код через якийсь час можна буде декомпілювати, проте обфускація може суттєво уповільнити процес декомпіляції

Для впровадження даного підходу було обрано продукт Obfuscar [28]. Це безкоштовний продукт, з відкритим кодом, який активно оновлюється та актуалізується спільнотою.

Основний принцип роботи даного обфускатор полягає в тому, що він перейменовує метадані в збірках .NET (наприклад, перейменовує методи, властивості, події, поля, типи та простори імен), використовуючи для назв максимально схожі імена для підвищення складності ідентифікації назв.

У нашому проекті, ми будемо використовувати версію 2.x. Під капотом, даний обфускатор використовує бібліотеку Mono.Cecil. Ця бібліотека спрощує роботу зі збірками та IL-кодом всередині них. Окремо варто зазначити, що Obfuscar підтримує як .NET Framework так і .NET Core.

Розглянемо принцип роботи обфускатора на прикладі бібліотеки ResearchBuy.Application.dll. Для декомпіляції коду будемо використовувати продукт з відкритим кодом ILSpy. На рис. 3.5 зображено приклад декомпіляції методу у збірці ResearchBuy.Application.dll. Декомпіляція виявилась досить простою.

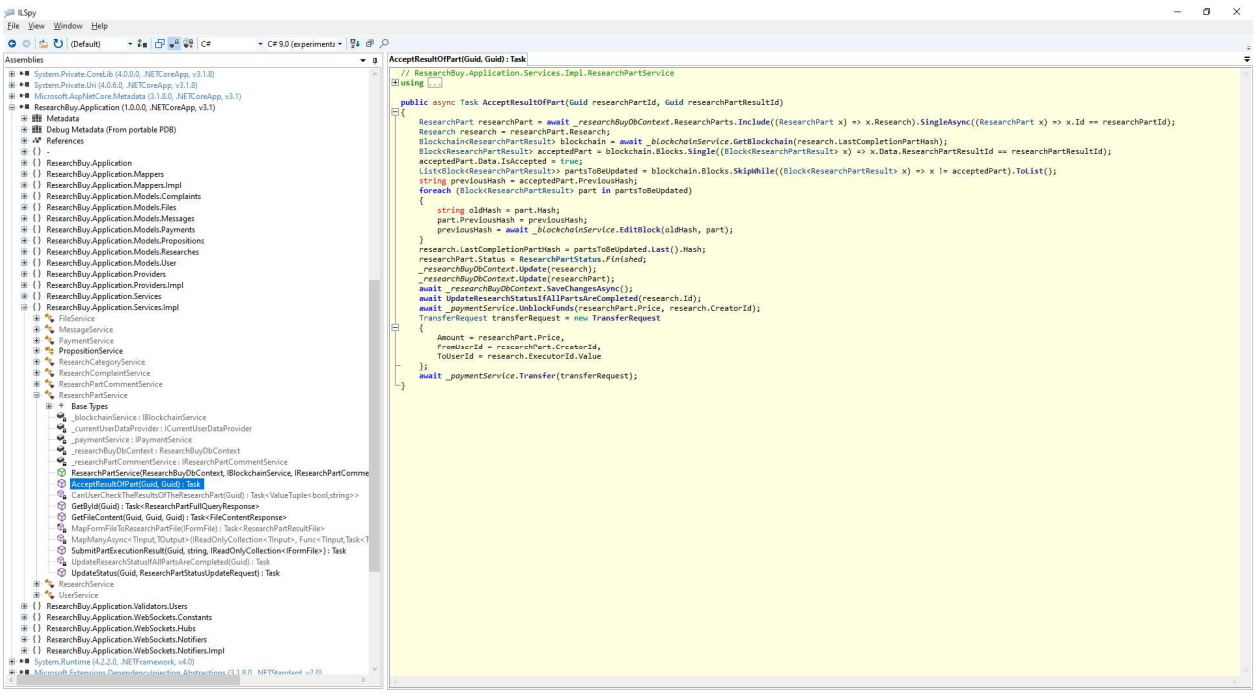


Рис. 3.5. Вміст методу AcceptResultOfPart класу ResearchPartService

Для того, щоб провести обфускацію продукту потрібно встановити Obfuscar. Зробити це можна за допомогою наступної команди: «dotnet tool install --global Obfuscar.GlobalTool --version 2.2.28».

Після установки, потрібно створити файл з налаштуваннями обфускації з назвою obfuscar.xml. На рис. 3.6 можна побачити приклад його вмісту.

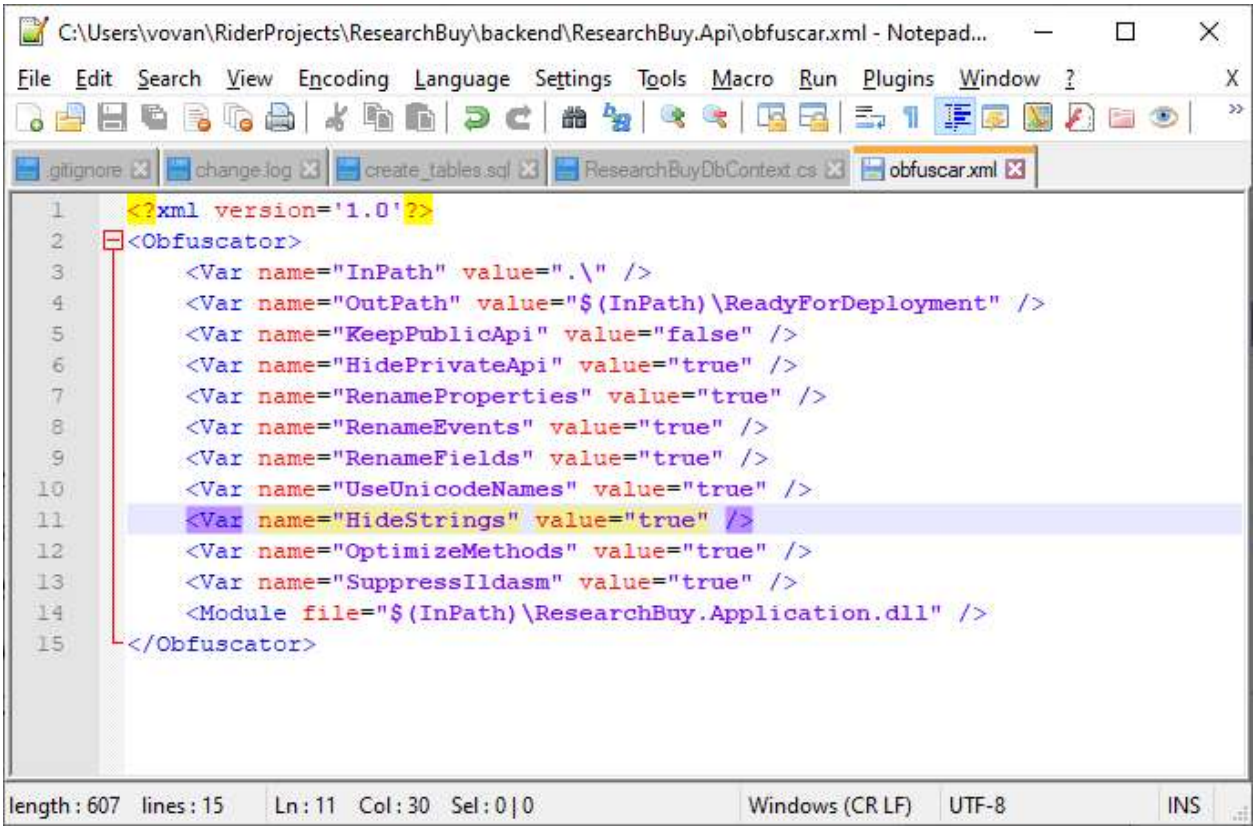


Рис. 3.6. Вміст файлу obfuscar.xml

Також потрібно налаштувати файл проекту, щоб після виконання побудови проекту відбувалось створення файлу з обфускованим вмістом.

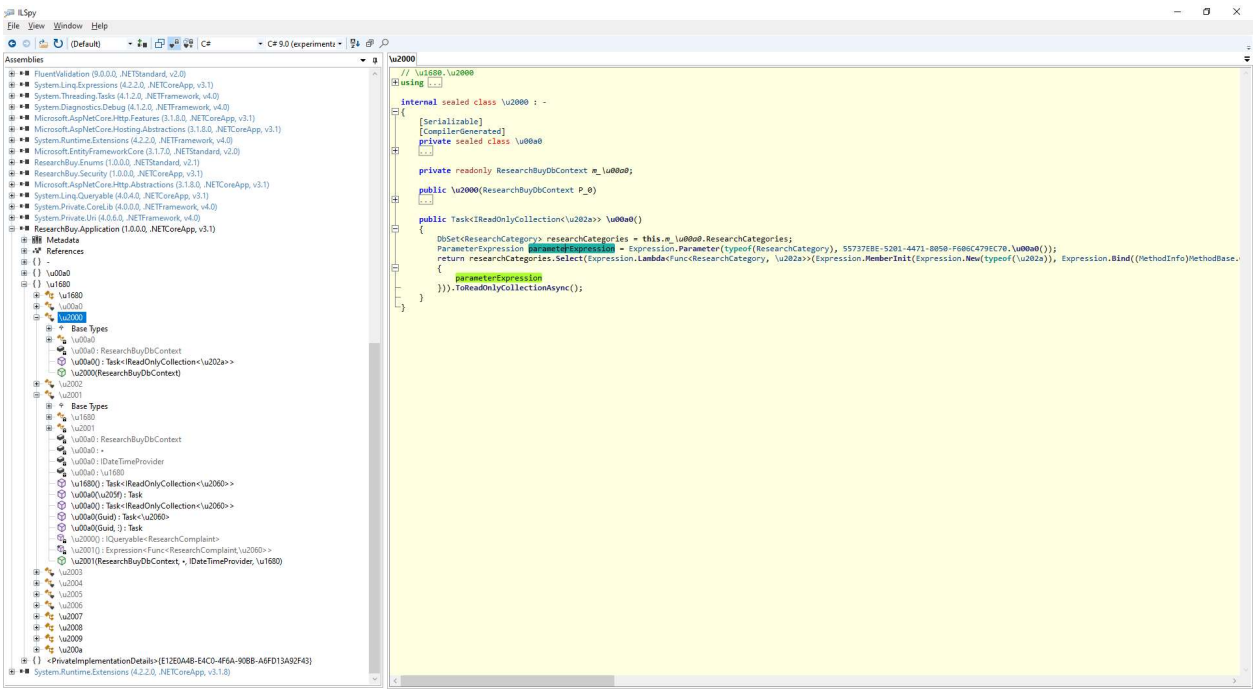


Рис. 3.7. Вміст збірки після використання Obfuscar

На рис. 3.7 можна побачити результат обфускації збірки. Власне, як і заявляли розробники – назви класів, методів, властивостей та полів змінились на більш менш однакові символи в Unicode. З мінусів, варто зазначити, що типи, які підключаються зовні збірки таки можна розпізнати і по контенту всередині методів можна розпізнати певні основні фази. Проте, на мою думку, тут ще стоїть питання ціни, адже даний обфускатор є безкоштовним. Хороше рішення на ринку для вирішення проблем такого роду може коштувати досить дорого.

ВИСНОВКИ ДО РОЗДІЛУ 3

У третьому розділі ми розглянули методи захисту, які були реалізовані в магістерській дипломній роботі. У ході імплементації додатку було вирішено впровадити:

- Механізми авторизації/аутентифікації
- Захищене та розподілене файлове сховище IPFS
- Захищений інструмент для збереження секретів Vault
- Захист вихідних файлів побудови додатку від декомпіляції засобом Obfuscar

Механізми аутентифікації/авторизації використовуються для визначення особистості, яка хоче виконати дію, і перевірки чи вона може здійснити цю дію. Дані механізми реалізуються в додатку стандартними засобами, які надає ASP.NET Core.

Для забезпечення захищеного збереження результатів виконання досліджень було використано IPFS. Цей інструмент надає можливості для зручного та надійного збереження файлів у розподіленій файловій системі.

Одним з місць, які завжди потребують захисту є секрети додатку. В нашому випадку було вирішено обрати Hashicorp Vault. За допомогою даного додатку, ми маємо можливість зберігати секрети у віддаленому місці у зашифрованому вигляді, що завадить зловмиснику, отримавши код проекту, дізнатись про них.

У рамках магістерської дипломної роботи було вирішено дослідити наявні механізми обфускації для .NET. Після дослідження наявних безкоштовних рішень на ринку було обрано Obfuscar. Приклади роботи з ним наведено в розділі 3.4.

РОЗДІЛ 4.

РЕАЛІЗАЦІЯ ТА ЕКСПЛУАТАЦІЯ ДОДАТКУ ДЛЯ ЗАХИЩЕНОГО ТА ФРАГМЕНТАРНОГО ПРОДАЖУ ДОСЛІДЖЕНЬ

У даному розділі буде зроблено огляд реалізації та експлуатації додатку для захищеного та фрагментарного продажу досліджень. Посторінково буде оглянуто основний функціонал розробленого програмного забезпечення, оглянуто основні шляхи API на серверній стороні, описано основні сутності БД, які беруть участь в обробці конкретного шляху.

4.1. Огляд реалізації рішення

У розділі 4.1 розглянемо реалізацію кожного з рівнів розробленої системи та детально зупинимось на кожному з компонентів додатку.

4.1.1. Реалізація рівню бази даних

Розпочнемо з опису схеми бази даних. В нашому додатку, в якості основної БД використовується SQL Server. У наступних таблицях буде детально представлено та описано вміст кожної з них.

У табл. 4.1 міститься опис таблиці користувачів. В ній зберігаються загальні дані про користувача та дані для фінансових операцій: поточний баланс та доступні кошти.

Таблиця 4.1

Користувачі (dbo.Users)

Назва	Тип	Опис
Id	INT	Унікальний ідентифікатор користувача

Email	NVARCHAR	Адреса електронної пошти
FullName	NVARCHAR	Повне ім’я
Password	NVARCHAR	Хеш паролю
PhoneNumber	NVARCHAR	Контактний номер телефону
DateOfBirth	DATETIME2	Дата народження
Role	INT	Роль в додатку (1 – звичайний користувач, 2 - адміністратор)
Balance	DECIMAL	Поточний баланс
AvailableFunds	DECIMAL	Доступні кошти
IsBlocked	BIT	Бітовий прапорець, для позначення користувача заблокованим.

У табл. 4.2 зберігаються дані по фінансовим транзакціям користувача. В них входить сума, тип та статус транзакції. Вони бувають наступного типу: 0 – поповнення, 1 – зняття, 2 – трансфер коштів між гаманцями. Також варто відзначити, що транзакція може мати 3 статуси: в процесі обробки (Pending), успішна (Success) та провалена (Failed).

Таблиця 4.2

Фінансові транзакції (dbo.Transactions)

Назва	Тип	Опис
Id	UNIQUEIDENTIFIER	Унікальний ідентифікатор транзакції
Description	NVARCHAR	Опис транзакції
Amount	DECIMAL	Сума транзакції
CreatedAt	DATETIME2	Дата створення
Type	INT	Тип транзакції
Status	INT	Статус транзакції
UserId	INT	Ідентифікатор користувача

В даній таблиці (табл. 4.3) зберігаються дані по файлам, які завантажуються в систему.

Таблиця 4.3

Файли (dbo.Files)

Назва	Тип	Опис
Id	UNIQUEIDENTIFIER	Унікальний ідентифікатор файлу
Name	NVARCHAR	Назва файлу
Path	NVARCHAR	Шлях до файлу
CreatedAt	DATETIME2	Дата створення

У таблиці 4.4 містить інформація про файли, які використовуються для показу аватарів користувачів. Таблиця зберігає зв’язок між користувачем та файлом, який використовується для відображення на вікні профілю.

Таблиця 4.4

Таблиця аватарів профілів користувачів (dbo.UserAvatarFiles)

Назва	Тип	Опис
Id	UNIQUEIDENTIFIER	Унікальний ідентифікатор
UserId	INT	Ідентифікатор користувача
FileId	UNIQUEIDENTIFIER	Ідентифікатор файлу

У таблиці 4.5 зберігаються основні категорії, за якими можуть проводитись дослідження.

Таблиця 4.5

Категорії досліджень (dbo.ResearchCategories)

Назва	Тип	Опис
Id	INT	Унікальний ідентифікатор категорії досліджень
Name	NVARCHAR	Назва

У таблиці 4.6 описано дані, які зберігаються в таблиці досліджень. Статус дослідження може мати наступні значення: створене (Created), в процесі виконання (InProgress), виконане (Completed) і провалене (Failed).

Таблиця 4.6

Дослідження (dbo.Researches)

Назва	Тип	Опис
Id	UNIQUEIDENTIFIER	Унікальний ідентифікатор дослідження
Header	NVARCHAR	Заголовок дослідження
Description	NVARCHAR	Опис дослідження
CreatedAt	DATETIME2	Дата створення
Status	INT	Статус дослідження
CreatorId	INT	Ідентифікатор замовника дослідження
ResearchCategoryId	INT	Ідентифікатор категорії дослідження
ExecutorId	INT	Ідентифікатор виконавця
TotalPrice	DECIMAL	Загальна ціна
LastCompletionPartHash	NVARCHAR	Хеш виконання останньої частини дослідження

У таблиці 4.7 зберігаються дані про частину дослідження. В ній міститься посилання на дослідження, а також є необв’язкове поле, яке містить посилання на частину пропозиції. Це зроблено для того, щоб не дублювати дані у різних таблицях. Також в даній таблиці міститься поле, яке визначає порядок у межах певної частини в дослідженні.

Таблиця 4.7

Частини досліджень (dbo.ResearchParts)

Назва	Тип	Опис
Id	UNIQUEIDENTIFIER	Унікальний ідентифікатор частини дослідження
Name	NVARCHAR	Назва частини
Description	NVARCHAR	Опис частини
Status	INT	Статус
ResearchId	UNIQUEIDENTIFIER	Ідентифікатор дослідження
CreatorId	INT	Ідентифікатор власника частини дослідження
Order	INT	Порядок частини в дослідженні
Price	DECIMAL	Ціна
PropositionResearchPartId	UNIQUEIDENTIFIER	Ідентифікатор на частину пропозиції

У таблиці 4.8 міститься опис схеми даних таблиці коментарів до частин досліджень. Вони необхідні для комунікації між виконавцем і власником дослідження.

Таблиця 4.8

Коментарі до частин досліджень (dbo.ResearchPartComments)

Назва	Тип	Опис
Id	UNIQUEIDENTIFIER	Унікальний ідентифікатор коментаря до частини дослідження
Content	NVARCHAR	Вміст коментаря
CreatedAt	DATETIME2	Дата створення
ResearchPartId	UNIQUEIDENTIFIER	Ідентифікатор частини дослідження
UserId	INT	Ідентифікатор користувача

У таблиці 4.9 зберігається опис даних необхідних для забезпечення механізму прикріплень файлів до досліджень. Відповідно до поставленого завдання, таблиця містить посилання на таблицю досліджень та файлів.

Таблиця 4.9

Прикріплені файли до досліджень (dbo.ResearchFiles)

Назва	Тип	Опис
Id	UNIQUEIDENTIFIER	Унікальний ідентифікатор
ResearchId	UNIQUEIDENTIFIER	Ідентифікатор дослідження
FileId	UNIQUEIDENTIFIER	Ідентифікатор файлу

У таблиці 4.10 можна побачити схему таблиці для збереження відгуків про виконання дослідження. Очікувано, таблиця містить посилання на дослідження, оцінку та вміст відгуку.

Таблиця 4.10

Відгуки про виконання досліджень (dbo.ResearchReviews)

Назва	Тип	Опис
Id	UNIQUEIDENTIFIER	Унікальний ідентифікатор відгуку про виконання дослідження
Mark	REAL	Оцінка (1-5)
Content	NVARCHAR	Вміст відгуку
ResearchId	UNIQUEIDENTIFIER	Ідентифікатор дослідження
ExecutorId	INT	Ідентифікатор виконавця
CreatedAt	DATETIME2	Дата створення

У таблиці 4.11 описано схему таблиці в якій зберігаються скарги, які виникли у користувачів в процесі виконання дослідження.

Таблиця 4.11

Скарги на виконання досліджень (dbo.ResearchComplaints)

Назва	Тип	Опис
Id	UNIQUEIDENTIFIER	Унікальний ідентифікатор скарги
Status	INT	Статус
Content	NVARCHAR	Вміст скарги
ResearchId	UNIQUEIDENTIFIER	Ідентифікатор дослідження
CreatorId	INT	Ідентифікатор користувача, який створив скаргу
CreatedAt	DATETIME2	Дата створення

У таблиці 4.12 зберігаються дані про коментарі до скарг. Через них адміністратори веб-сайту та користувачі можуть взаємодіяти між собою.

Таблиця 4.12

Коментарі до скарг (dbo.ResearchComplaintComments)

Назва	Тип	Опис
Id	UNIQUEIDENTIFIER	Унікальний ідентифікатор коментарю в скарзі
ResearchComplaintId	UNIQUEIDENTIFIER	Ідентифікатор скарги
Content	NVARCHAR	Вміст скарги
UserId	INT	Ідентифікатор користувача, який створив коментар
CreatedAt	DATETIME2	Дата створення

У таблиці 4.13 містяться дані, які використовуються для представлення пропозиції по виконанню певного завдання. Вона містить посилання на дослідження та ідентифікатор користувача, який створив пропозицію.

Таблиця 4.13

Пропозиції на виконання досліджень (dbo.Propositions)

Назва	Тип	Опис
Id	UNIQUEIDENTIFIER	Унікальний ідентифікатор пропозиції
Status	INT	Статус
ResearchId	UNIQUEIDENTIFIER	Ідентифікатор дослідження
UserId	INT	Ідентифікатор користувача
CreatedAt	DATETIME2	Дата створення

У таблиці 4.14 описані дані для зв’язування пропозиції з окремими частинами, які були створенні при представленні пропозиції для власника завдання.

Таблиця 4.14

Частини пропозицій на виконання дослідження

(dbo.PropositionResearchParts)

Назва	Тип	Опис
Id	UNIQUEIDENTIFIER	Унікальний ідентифікатор
PropositionId	UNIQUEIDENTIFIER	Ідентифікатор пропозиції
ResearchPartId	UNIQUEIDENTIFIER	Ідентифікатор частини дослідження

У таблиці 4.15 міститься зв’язка пропозиції з прикріпленими файлами. Так як, потенційний виконавець, може пропонувати свої варіанти рішень, то було вирішено створити окрему таблицю для зв’язування пропозицій з завантаженими файлами.

Таблиця 4.15

Файли прикріплені до пропозицій (dbo.PropositionFiles)

Назва	Тип	Опис
Id	UNIQUEIDENTIFIER	Унікальний ідентифікатор
PropositionId	UNIQUEIDENTIFIER	Ідентифікатор пропозиції
FileId	UNIQUEIDENTIFIER	Ідентифікатор файлу

У таблиці 4.16 зберігаються дані по коментарям, які можуть бути доданими для певної пропозиції. (Не забути при форматуванні поміняти порядок).

Таблиця 4.16

Коментарі до пропозицій (dbo.PropositionComments)

Назва	Тип	Опис
Id	UNIQUEIDENTIFIER	Унікальний ідентифікатор коментарю в пропозиції
PropositionId	UNIQUEIDENTIFIER	Ідентифікатор пропозиції

UserId	INT	Ідентифікатор користувача, який створив пропозицію
Content	NVARCHAR	Вміст коментарю
CreatedAt	DATETIME2	Дата створення

4.1.2. Реалізація серверного рівню

Для реалізації серверного веб-додатку було обрано платформу ASP.NET Core. Серверний додаток являє собою Web API розроблене за підходом REST. В певних випадках для підвищення інтерактивності додатку було використано технологію для роботи з веб-сокетами SignalR. В ході розробки було вирішено поділити його на певні проекти, кожен з яких матиме чітку відповідальність. Проведемо огляд функціоналу та призначень даних проектів.

4.1.2.1. Огляд проекту ResearchBuy.Configuration

У проекті ResearchBuy.Configuration зберігаються класи для роботи з налаштуваннями та секретами додатку.

В додатку є 2 варіанти для взяття конфігурацій – appsettings.json файл та Hashicorp Vault. Перший варіант – це звичайний незахищений файл, стандартний спосіб для збереження конфігурацій в ASP.NET Core. Другий варіант – це захищене та спеціалізоване програмне забезпечення для збереження секретів додатку.

На даний момент, в проекті зберігаються наступні конфігурації:

- DbSettings – відповідає за налаштування з’єднання до бази даних SQL Server.
- AuthSettings – містить налаштування, які використовуються після успішної аутентифікації користувача для створення JWT-токена.

4.1.2.2. Огляд проекту ResearchBuy.Enums

Оглянемо наступний проект ResearchBuy.Enums. Він використовується для збереження перерахувань, які використовуються в додатку. Коротко ознайомимось з вмістом даного проекту та існуючими перерахуваннями.

Таблиця 4.17

Існуючі перерахування

Назва перерахування	Рядкове значення	Числова репрезентація	Опис
ComplaintStatus	Pending	0	Початковий стан скарги
	InProgress	1	Скарга в процесі розгляду
	Accepted	2	Скарга визначена як справедлива, вимоги позивача задовільнені
	Rejected	3	Скарга відхилена
PropositionStatus	Pending	1	Пропозиція на стадії розглядання
	Accepted	2	Пропозиція прийнята
	Declined	3	Пропозиція відхилена
ResearchPartStatus	NeedsFunds	0	Потрібно внести гроші на баланс гаманця, щоб виконавець міг приступити до виконання завдання
	Pending	1	Гроші внесені, частина готова до того, щоб бути взятою в обробку
	InProgress	2	Частина в процесі виконання

	OnReview	3	Частина знаходиться на стадії перевірки
	Finished	4	Частина закінчена
	Failed	5	Частина провалена
ResearchStatus	Created	1	Щойно створене дослідження (ще не взяте в роботу)
	InProgress	2	Дослідження в розробці
	Completed	3	Успішно завершене дослідження
	Failed	4	Провалене дослідження
Role	Standard	1	Роль звичайного користувача
	Admin	2	Роль адміністратора веб-сайту
TransactionStatus	Pending	1	Транзакція в обробці
	Success	2	Успішна транзакція
	Failed	3	Провалена транзакція
TransactionType	Deposit	0	Транзакція для поповнення коштів
	Withdraw	1	Транзакція для зняття коштів з рахунку
	Transfer	2	Транзакція для переводу грошей з рахунку на рахунок

4.1.2.3. Огляд проекту ResearchBuy.Persistence

Проект ResearchBuy.Persistence відповідає за роботу з базою даних SQL Server. Робота ведеться через бібліотеку для роботи з реляційними БД Entity Framework Core [29].

Дана бібліотека забезпечує роботу технологію ORM. Розглянемо детальніше дане поняття. ORM – це техніка для роботи з базою даних, при якій ми працюємо з нею через певну обгортку, яка бере на себе значну частину роботи з генерування запитів та конвертації повернених результатів у відповідні об’єкти в нашому коді на C# або на іншій мові програмування. Це дозволяє зекономити час розробки, при розростанні додатку – зменшити час необхідний на підтримку існуючих SQL-запитів.

Для забезпечення конвертації даних повернених з БД при виконанні SQL-запитів нам потрібні моделі. В нашому проекті, модель – це репрезентація таблиці, яка знаходиться на стороні БД. Відповідно, в папці Models містяться моделі, які однозначно відповідають таблицям на стороні БД. Для конфігурації специфічних зв’язків або обмежень можна використовувати імплементації інтерфейсу IEntityTypeConfiguration<T>, за допомогою яких прописувати зв’язки, ключі, обмеження, які існують на стороні БД.

Центральною сутністю для роботи з рівнем бази даних є її контекст. Для цього було створено клас ResearchBuyDbContext. Він реалізує стандартний абстрактний клас DbContext. По суті, цей контекст є відображенням БД у нашому коді в C#. Для того, щоб мати доступ до певної таблиці необхідно додати властивість DbSet<T>, де T – це тип моделі, а ім’я властивості – це назва таблиці. За замовчуванням, використовується стандартна схема dbo. Після додавання властивості, ми можемо в інших класах використовувати її для роботи з певною таблицею через механізм LINQ, який конвертує код в C# у відповідні запити SQL. Також варто відмітити, що для реєстрації специфічних зв’язків та обмежень можна використати метод OnModelCreating.

Бібліотека Entity Framework Core містить вбудований механізм міграцій. Це означає, що ми можемо спочатку створити або модифікувати об’єкти в коді C#, а потім на основі цих змін буде автоматично змінена схема БД. У нашому проекті міграції зберігаються в папці Migrations.

4.1.2.4. Огляд проекту ResearchBuy.Application

Проект ResearchBuy.Application є місцем, де сконцентрована бізнес-логіка додатку. Більшість логіки міститься в сервісах. Зазвичай, вони працюють наступним чином. При взаємодії з клієнтською частиною викликаються певні методи в сервісах. На основі вхідних даних HTTP-запиту, ми проводимо певні дії з даними (перетворення, валідацію, зчитування) та записуємо їх в базу даних. У Д2.1 міститься перелік існуючих сервісів та їх функціонал.

4.1.2.5. Огляд проекту ResearchBuy.Api

Залишилось розглянути тільки останній проект – ResearchBuy.Api. Головним завданням даного проекту є надання можливості для виклику бізнес-логіки через HTTP. Для цього використовується бібліотека ASP.NET Core. Вона дозволяє у зручній та ефективній формі створювати обробники для них, які, зазвичай, знаходяться у контролерах. У таблиці 4.19 перераховані існуючі контролери у рішенні.

Таблиця 4.19

Контролери

Назва контролеру	Метод	URL	Опис
File Controller	POST	api/v1/files	Створення та запис файлів, які були передані з HTTP-запитом
	GET	api/v1/files/{fileId}	Зчитування вмісту файлу

Payments Controller	GET	api/v1/payments/balance	Зчитування балансу поточного користувача
	POST	api/v1/payments/deposit	Здійснення внесення коштів на рахунок поточного користувача
	POST	api/v1/payments/withdrawal	Здійснення виведення коштів з рахунку поточного користувача
	POST	api/v1/payments/transfer	Здійснення переводу коштів між гаманцями користувачів
Proposition Controller	POST	api/v1/propositions	Створення нової пропозиції по дослідженню
	GET	api/v1/propositions	Зчитування пропозицій, які зробив поточний користувач
	GET	api/v1/propositions/ {propositionId}	Зчитування пропозиції за заданим ідентифікатором
	POST	api/v1/propositions/{proposition Id}/accept	Прийняття пропозиції за даним ідентифікатором
	POST	api/v1/propositions/comments	Створення коментарю для певної пропозиції
Research Category Controller	GET	api/v1/researches/categories	Зчитування категорій досліджень, які існують в системі
Research Complaint Controller	GET	api/v1/researchComplaints/all	Зчитування усіх скарг
	GET	api/v1/researchComplaints/my	Зчитування скарг, які були створені виключно поточним користувачем

	GET	api/v1/researchComplaints/{complaintId}	Зчитування скарги за ідентифікатором
	PUT	api/v1/researchComplaints/{complaintId}/status	Модифікація статусу скарги
	POST	api/v1/researchComplaints	Створення нової скарги
	POST	api/v1/researchComplaints/{complaintId}/comments	Створення коментарю в скарзі
Research Controller	POST	api/v1/researches	Створення нового дослідження
	GET	api/v1/researches	Зчитування усіх досліджень створених поточним користувачем
	GET	api/v1/researches/orders	Зчитування досліджень, у яких поточний користувач є викоанавцем
	POST	api/v1/researches/find	Пошук досліджень за ціною, назвою та категорією
	GET	api/v1/researches/{researchId}	Зчитування дослідження за даним ідентифікатором
	GET	api/v1/researches/{researchId}/reviews	Зчитування відгуку для дослідження
	POST	api/v1/researches/{researchId}/reviews	Створення нового відгуку про виконання дослідження
	GET	api/v1/researches/{researchId}/status	Зчитування статусу дослідження
	POST	api/v1/researches/{researchId}/f	Закінчення дослідження

		ail	власником у випадку незадовільних результатів
Research Part Controller	GET	api/v1/researchParts/{researchPartId}	Зчитування частини дослідження за заданим ідентифікатором
	POST	api/v1/researchParts/{researchPartId}/results	Завантаження результатів виконання частини дослідження на сервер
	GET	api/v1/researchParts/{researchPartId}/results/{researchPartResultId}/files/{fileId}	Зчитування файлу з результатів виконання частини дослідження
	PUT	api/v1/researchParts/{researchPartId}/results/{researchPartResultId}/	Успішне закінчення виконання частини дослідження
	PUT	api/v1/researchParts/{researchPartId}/status	Модифікація статусу частини дослідження
	POST	api/v1/researchParts/{researchPartId}/reserveFunds	Резервування коштів для виконання частини дослідження
	POST	api/v1/researchParts/{researchPartId}/comments	Створення коментарю до частини дослідження
User Controller	GET	api/v1/users	Зчитування усіх користувачів в системі
	POST	api/v1/users/register	Реєстрація нового користувача
	POST	api/v1/users/signIn	Проведення аутентифікації користувача

	GET	api/v1/users/{userId}	Зчитування профілю користувача за даним ідентифікатором
	PUT	api/v1/users/{userId}/block	Блокування користувача в системі
	PUT	api/v1/users/{userId}/unblock	Розблокування користувача в системі

У ході розробки рішення виникла проблема з забезпеченням оновлення сторінок з коментарями у режимі реального часу. Для цієї задачі було більш вигідно використати механізм веб-сокетів, так як він дозволяє надсилати оновлення з серверу напряму потрібним клієнтам. Типовим рішенням для роботи з веб-сокетами на платформі .NET Core є бібліотека SignalR.

Крім контролерів даний проект містить також файл Startup. Він відповідає за підключення всіх необхідних бібліотек та реєстрацію сервісів через механізм ін’єкції залежностей.

4.1.3. Реалізація клієнтського рівню

На клієнтському рівні було реалізовано однострінковий додаток на базі фреймворку Angular та базових засобів для створення веб-сторінок – HTML, CSS та JS. Як уже було написано в розділі 2.1.1, основним будівельним блоком у розробці клієнтського додатку є компонент. Таблиця огляду функціональності кожного з компонентів міститься у Д2.2.

Клієнтський додаток взаємодіє з серверною частиною через стандартну бібліотеку @angular/common/http. Вона надає зручну обгортку для виконання HTTP-запитів з різними типами контентів, підтримує вбудовану серіалізацію об’єктів у формат JSON. При потребі, є можливість завантаження файлів через форму.

Для роботи з JWT-токеном використовується сервіс AuthService. Після проходження етапу входу в додаток згенерований токен зберігається в локальне сховище. При потребі, з нього можна дістати таку корисну інформацію, як ідентифікатор користувача, адресу електронної пошти, повне ім’я та роль. При кожному наступному HTTP-запиті даний токен буде автоматично включений в запит за допомогою JwtInterceptor. Інтерсептор – це сутність в фреймворці Angular, яка дозволяє автоматично додавати певні дані при виконанні запитів [30].

4.2. Огляд експлуатації рішення

У даному розділі буде зроблено огляд реалізації додатку для захищеного та фрагментарного продажу досліджень. Посторінково буде оглянуто основний функціонал розробленого програмного забезпечення.

4.2.1. Реєстрація нового користувача

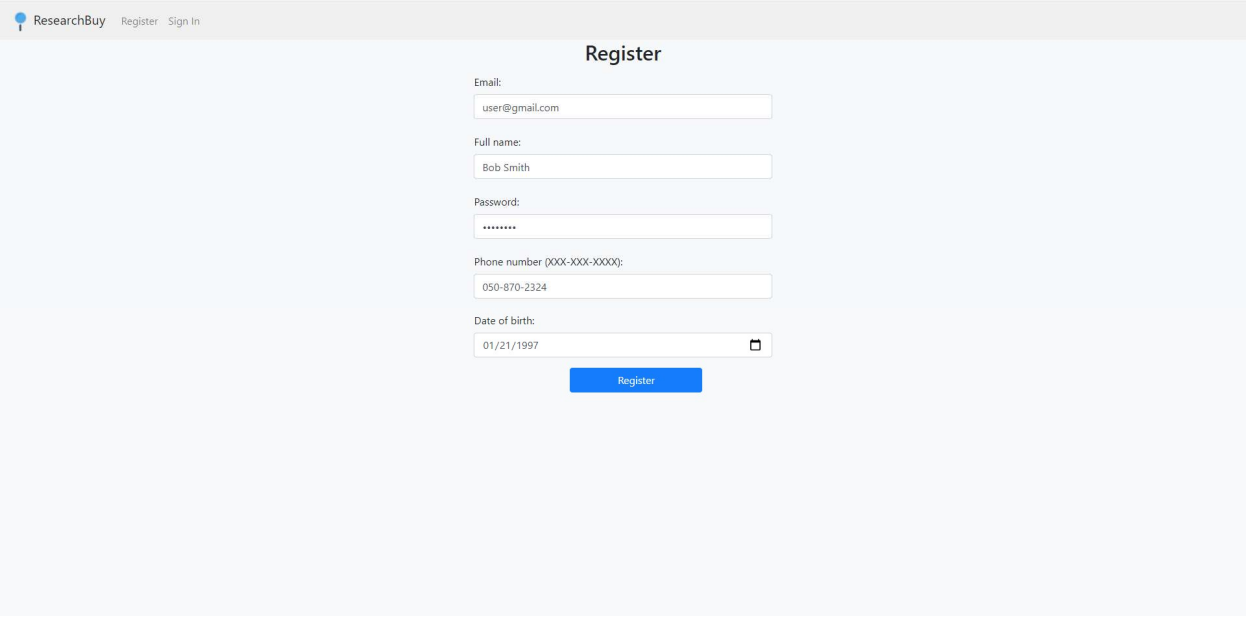


Рис. 4.1. Сторінка реєстрації нового користувача

Для початку користування додатком, потенційний користувач має створити новий обліковий запис (рис. 4.1). Щоб зробити це він має ввести дані про себе: адресу поштової скриньки, ім’я та прізвище, пароль, номер телефону та дату народження. Після введення всіх необхідних даних клієнтський додаток здійснює перевірку валідності введених даних.

Після натиснення кнопки «Register» користувач, клієнтський додаток надсилає запит на адресу api/v1/users/register. На сервері повторно здійснюється перевірка введених даних. Зокрема, здійснюється перевірка чи користувач з такою адресою електронної пошти був уже зареєстрований в додатку. Якщо такий користувач був зареєстрований, то сервер повертає помилку з повідомленням про те, що користувач з такою поштою уже існує в системі.

4.2.2. Вхід в додаток

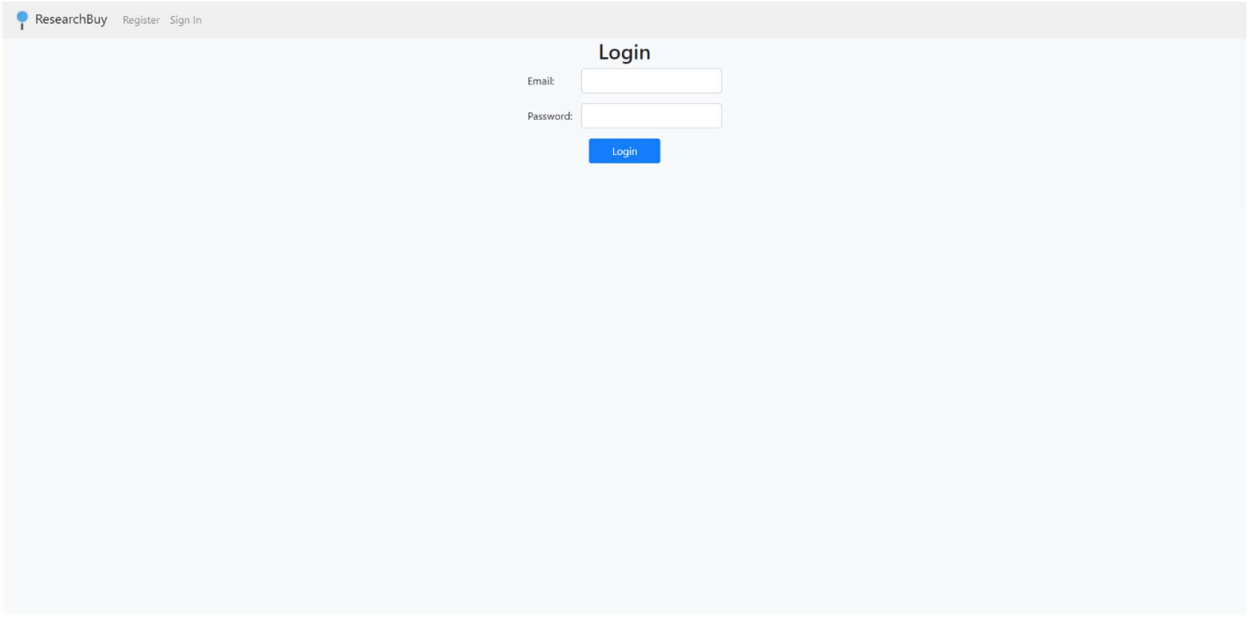


Рис. 4.2. Сторінка входу користувача в додаток

Після успішної реєстрації користувача в додатку, він має ввести адресу електронної скриньки та пароль, щоб потрапити на домашню сторінку веб-сайту. Коли користувач натискає на кнопку «Login» (рис. 4.2), клієнтський додаток надсилає запит на api/v1/users/signIn.

Якщо пошта та пароль співпадають з тими, які наявні в таблиці Users, то користувач має можливість зайти в додаток. Також при поверненні результату запиту, клієнтський додаток бере JWT-токен, який повертається у разі успішного входу в додаток. JWT-токен буде надалі використовуватись для механізмів аутентифікації та авторизації.

4.2.3. Створення нового дослідження

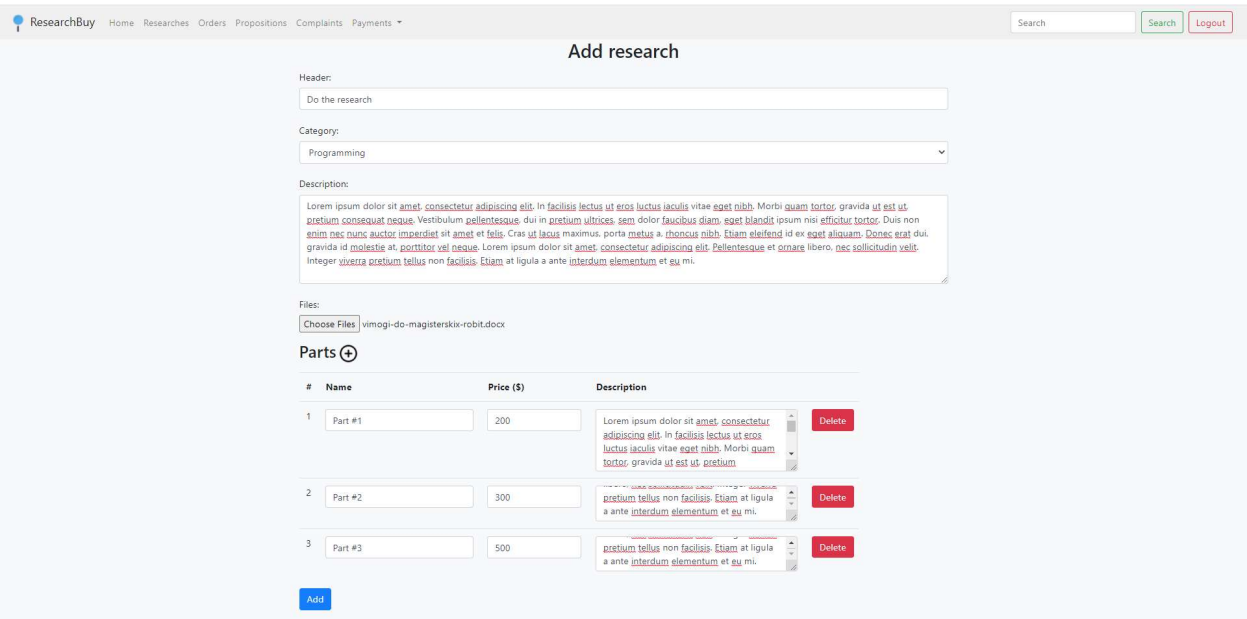


Рис. 4.3. Сторінка нового дослідження

На рис. 4.3 зображена сторінка для створення нового дослідження. На ній користувач має можливість ввести всі необхідні деталі:

- Заголовок
- Категорію
- Опис
- Завантажити прикріплення
- Розбити завдання на частини

При управлінні частинами, користувач, може додавати або видаляти частини. Кожна частина містить назву, ціну і опис. Загальна ціна дослідження буде дорівнювати сумі ціни всіх частин.

Після того, як користувач введе необхідні дані, клієнтський додаток надсилає запит за адресою api/v1/researches. Відповідно, після цього серверний додаток приймає запит і починає його обробку. Під час обробки ми додаємо запис в таблицю Research та створюємо записи для частин досліджень в таблицю ResearchPart. Після створення дослідження повертається відповідь з HTTP-статусом 200 (OK).

4.2.4. Сторінка пошуку досліджень для виконання

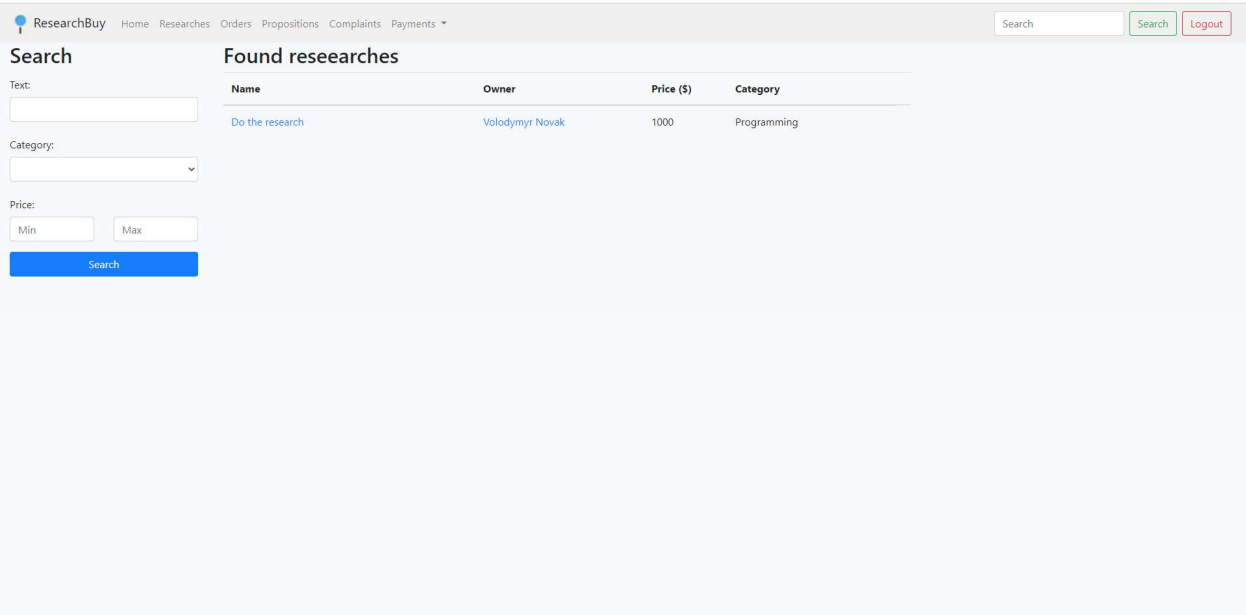


Рис. 4.4. Сторінка пошуку досліджень для виконання

Щоб ефективно здійснювати пошук досліджень, які можна виконати було розроблено окрему сторінку (рис. 4.4). Для переходу на неї достатньо ввести необхідний текст для пошуку і натиснути на кнопку «Search». На цій сторінці користувач має можливість відфільтрувати наявні дослідження за такими параметрами:

- Текст
- Категорія
- Діапозон цін

Після заповнення вищевказаних даних, клієнтський додаток надсилає запит за адресою api/v1/researches/find. Відповідно, до даних в запиті, здійснюється пошук необхідних даних в таблицях Research та ResearchPart. Пошук за текстом проводиться по таким полям як назва і опис дослідження, а також його частин.

Результати запиту відображаються в таблиці. В ній містяться дані про назву знайдених досліджень, власника дослідження, загальну ціну та категорії. Натиснувши на посилання, можна легко перейти на сторінку перегляду дослідження.

4.2.5. Сторінка перегляду поточного дослідження

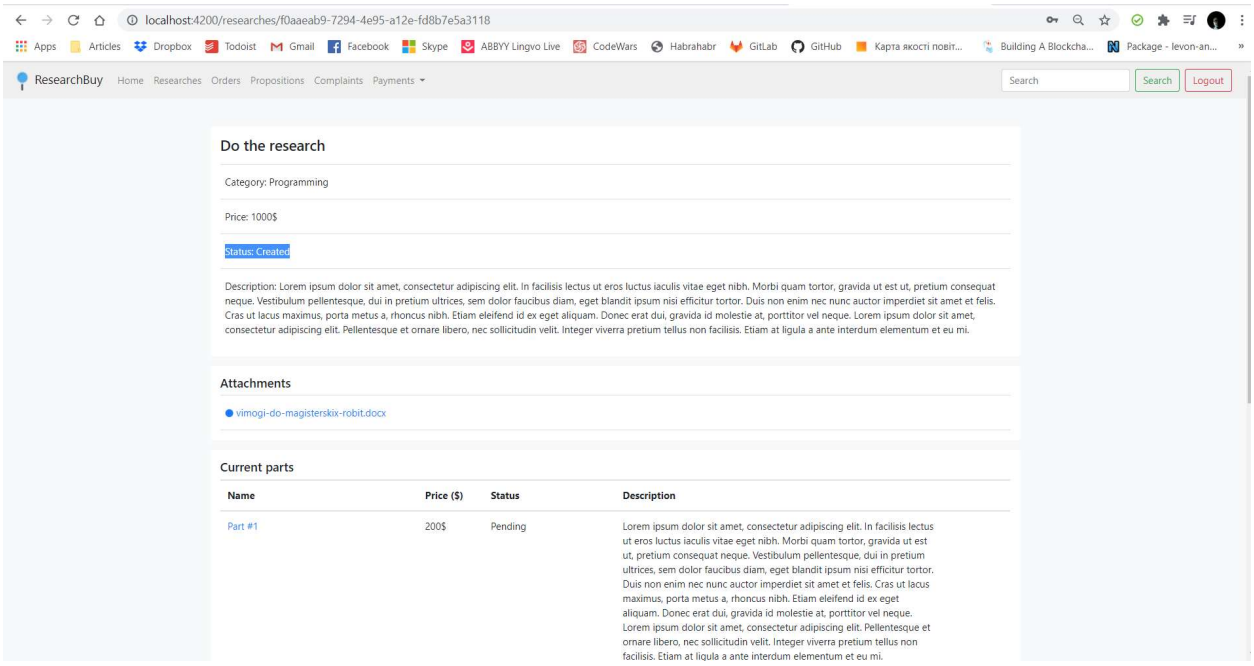


Рис. 4.5. Сторінка перегляду дослідження

Після створення дослідження, користувач може перейти на сторінку, де зможе переглянути його (рис. 4.5). На цій сторінці користувачу доступні дані, які він ввів при дослідженні: заголовок, категорія, ціна, опис та прикріплені файли, відображаються частини завдання, на яких є посилання за якими користувач може детальніше оглянути прогрес по виконанню конкретної частини.

Рис. 4.6. Таблиця з поточними пропозиціями по дослідженню

Також на даній сторінці, є список користувачів, які зробили пропозицію на виконання даного дослідження (рис. 4.6). Відповідно, власник дослідження може вибрати виконавця завдання зі списку тих людей, які зробили пропозицію. У цій частині додатку користувач може розглянути пропозиції, які поступили на виконання та натиснути на кнопку «Accept», у разі, коли він хоче прийняти певну пропозицію.

При натисненні даної кнопки клієнтський додаток надсилає запит api/v1/propositions/{propositionId}/accept. Обробник даного запиту копіює частини та оновлює загальну ціну для дослідження. Також до прикріплених файлів додаються ті, які потенційний виконавець завдання підвантажив у пропозицію.

Після прийняття пропозиції по дослідженню, на рахунку власника дослідження автоматично блокуються кошти для реалізації першої частини дослідження. При початку виконання кожної з частин, власник має натиснути на кнопку «Reserve Funds», яка зарезервує кошти для її виконання. Виконавець не може взяти частину у виконання, до тих пір поки власник не зарезервує кошти.

Якщо виконавця або власника дослідження є скарги на одну зі сторін, вони можуть зробити це, натиснувши на кнопку «Complaint». Якщо ж

виконавець хоче достроково завершити завдання, то він може натиснути на кнопку «Fail». Вона спрацює лише у тому разі, якщо у виконавця немає зараз частин завдання, які він виконує. В такому разі, необхідно писати скаргу адміністратору, щоб достроково припинити виконання дослідження з частиною, яка зараз є в прогресі.

4.2.6. Сторінка створення пропозиції для дослідження

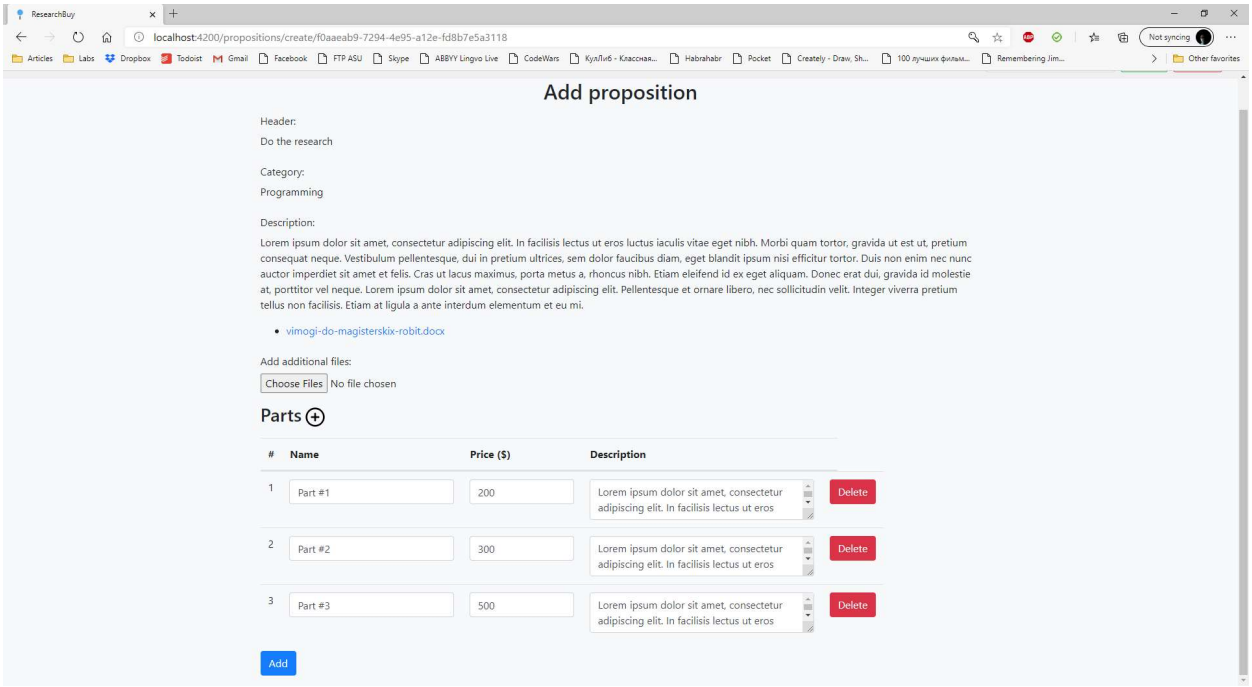


Рис. 4.7. Сторінка створення нової пропозиції по дослідженню

Для створення нової пропозиції, користувачу потрібно натиснути на кнопку «Make proposition» на сторінці дослідження, яке його зацікавило. Після цього користувач переходить на сторінку створення пропозиції по дослідженню. На ній відображаються загальні дані по дослідженню: заголовок, опис, категорія, прикріплені файли.

Варто відзначити, що на цій сторінці користувач має можливість редагувати частини, які надав власник завдання. Тобто людина, яка робить пропозицію може видалити частину або додати її, змінити вміст та ціну. Це дає

можливість краще розбити завдання на фрагменти, щоб краще оцінити зусилля та об’єм роботи для даного дослідження.

4.2.7. Сторінка перегляду частини дослідження

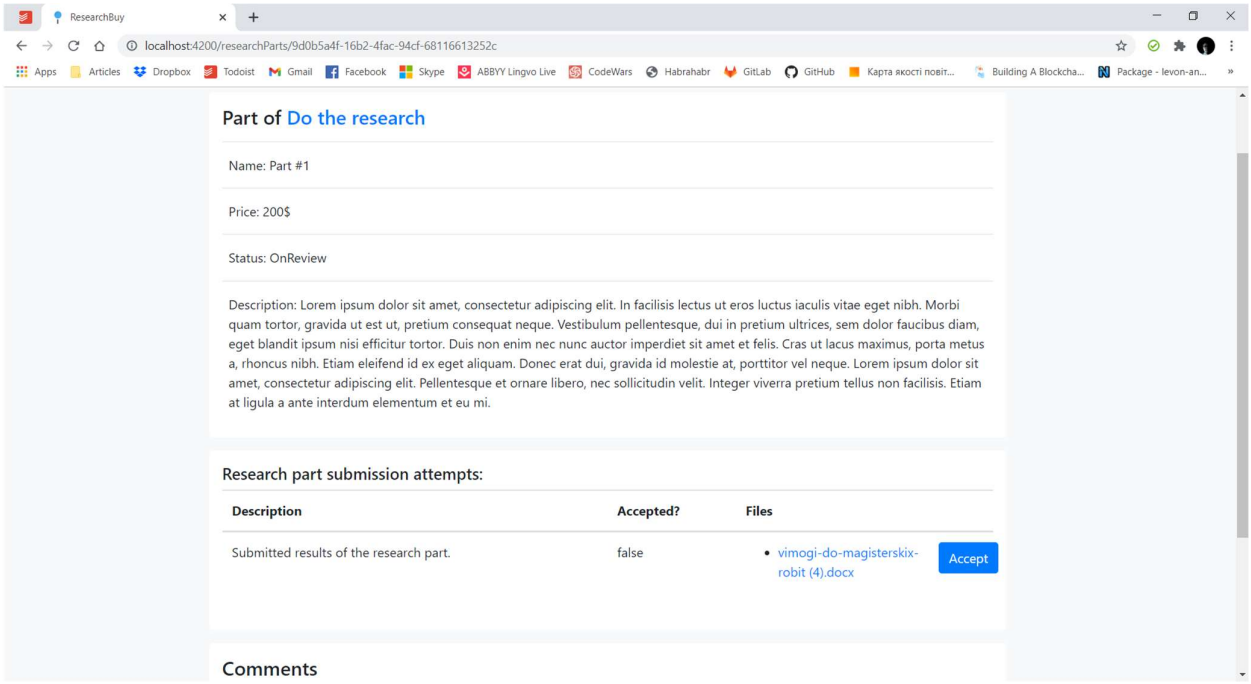


Рис. 4.8. Сторінка перегляду частини дослідження

На рис. 4.8 зображена сторінка відображення частини дослідження. На даній сторінці користувач може відстежувати результат виконання дослідження. Клієнтський веб-сайт надає можливість для перегляду інформації по вмісту дослідження: відображається назва частини, вміст, статус та опис.

Також у разі якщо на сторінці знаходиться виконавець, то є можливість завантаження файлів з результатами виконання дослідження на сервер. Після цього, власник дослідження може перевірити результати виконання дослідження і прийняти їх, якщо його влаштовують результати. Якщо ж у нього є певні конкретні пропозиції по правкам завдання, він може написати їх в коментарях до частини завдання.

Варто відмітити, що при натисненні кнопки «Accept» кошти за виконану частину завдання остаточно перейдуть з гаманця власника завдання до

виконавця. До цього моменту вони перебувають у заблокованому стані. У результаті частина завдання стає виконаною. Після виконання всіх частин завдання воно вважається виконаним.

4.2.8. Сторінка проведення платежів

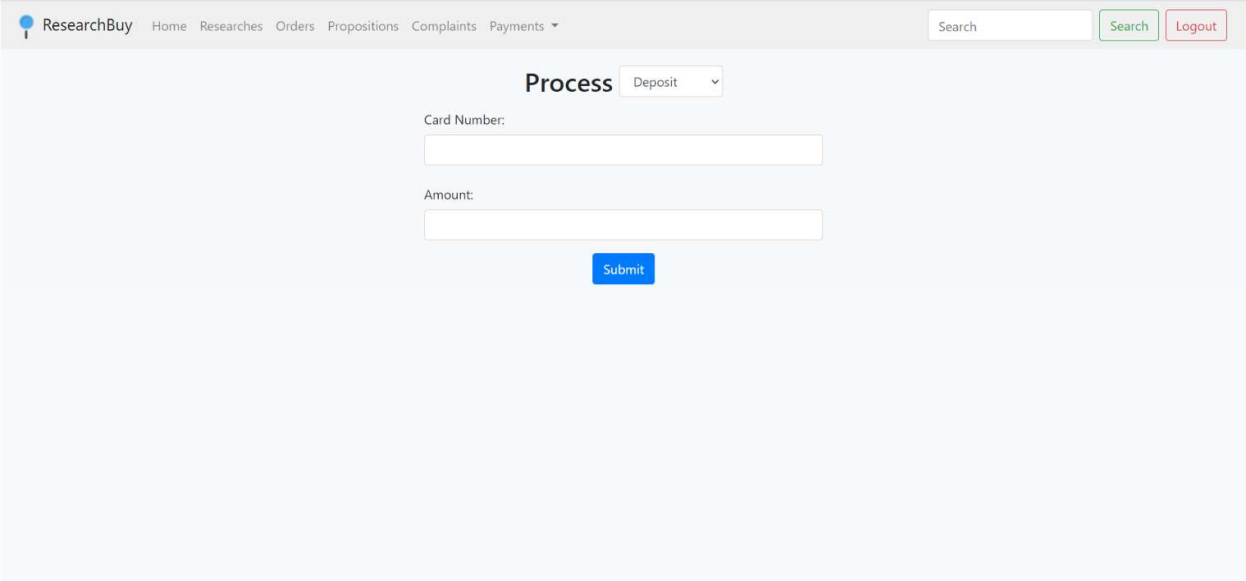


Рис. 4.9. Сторінка проведення платежів

Для здійснення зарахувань та здійснення виплат було вирішено створити сторінку для проведення платежів (рис. 4.9). На цій сторінці користувач може як вивести гроші зі свого рахунку (withdrawal), так і поповнити його (deposit). Для цього необхідно ввести номер картки та суму на яку має бути здійснена операція. По замовчуванню, всі суми в системі зараз проводяться в доларах США.

Поки, ця сторінка є лише заглушкою для підключення існуючої платіжної системи. Було зроблено спроби підключення PrivatBank API, проте, навіть для роботи в тестовому режимі потрібно було мати публічну адресу сайту.

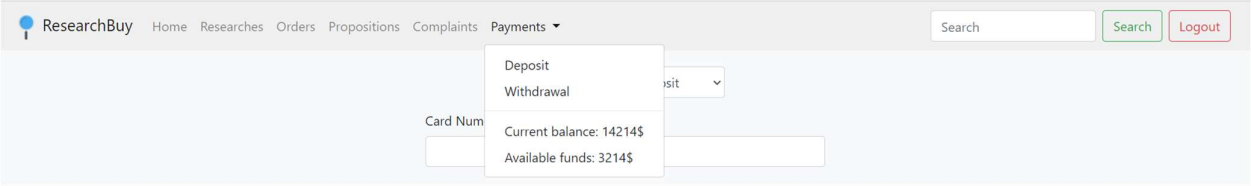


Рис. 4.10. Перегляд балансу поточного користувача

При натисненні на пункт меню «Payments», користувач має можливість переглянути власний баланс. В розробленій системі, існує 2 поняття – поточний баланс та доступні кошти. Доступні кошти – це поточний баланс з відніманням заблокованих коштів. При початку виконання завдання, власник має зарезервувати кошти для його виконання. Відповідно, після закінчення завдання кошти повністю переходять на гаманець виконавця. Даний механізм був введений для того, щоб зменшити ймовірність здійснення шахрайницьких дій з боку власника дослідження.

4.2.9. Сторінка створення скарги на учасника дослідження

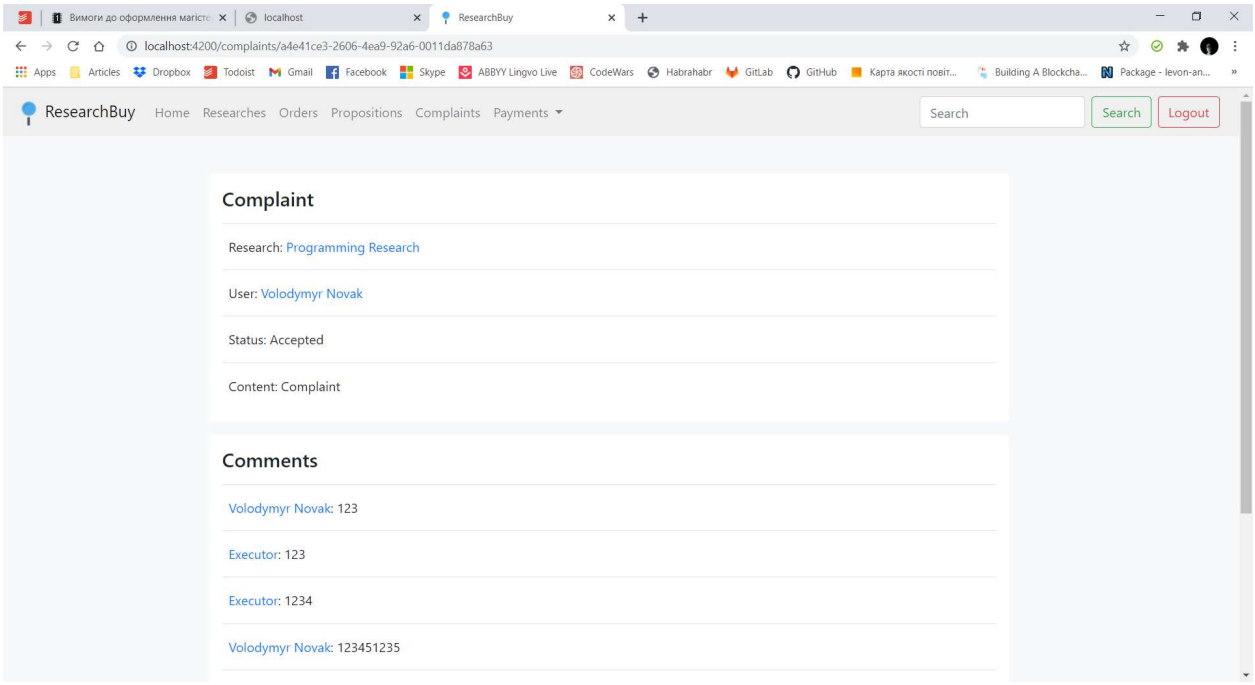


Рис. 4.11. Вікно перегляду скарги

На рис. 4.11 зображено вікно перегляду скарги. Якщо виконавця або замовника не влаштовує певна поведінка або дії користувача, то він має можливість залишити скаргу, вказавши дослідження до якого вона відноситься та її вміст.

Після подання скарги адміністратор має розглянути її та винести рішення. Для обговорення деталей, адміністратор має можливість залишити коментар по скарзі. При вирішенні спорів, адміністратор може переслати певну суму з одного аккаунту на інший, таким чином, вирішивши проблему, коли хтось не оплатив виконану роботу.

4.2.10. Сторінка створення відгуку про виконання дослідження

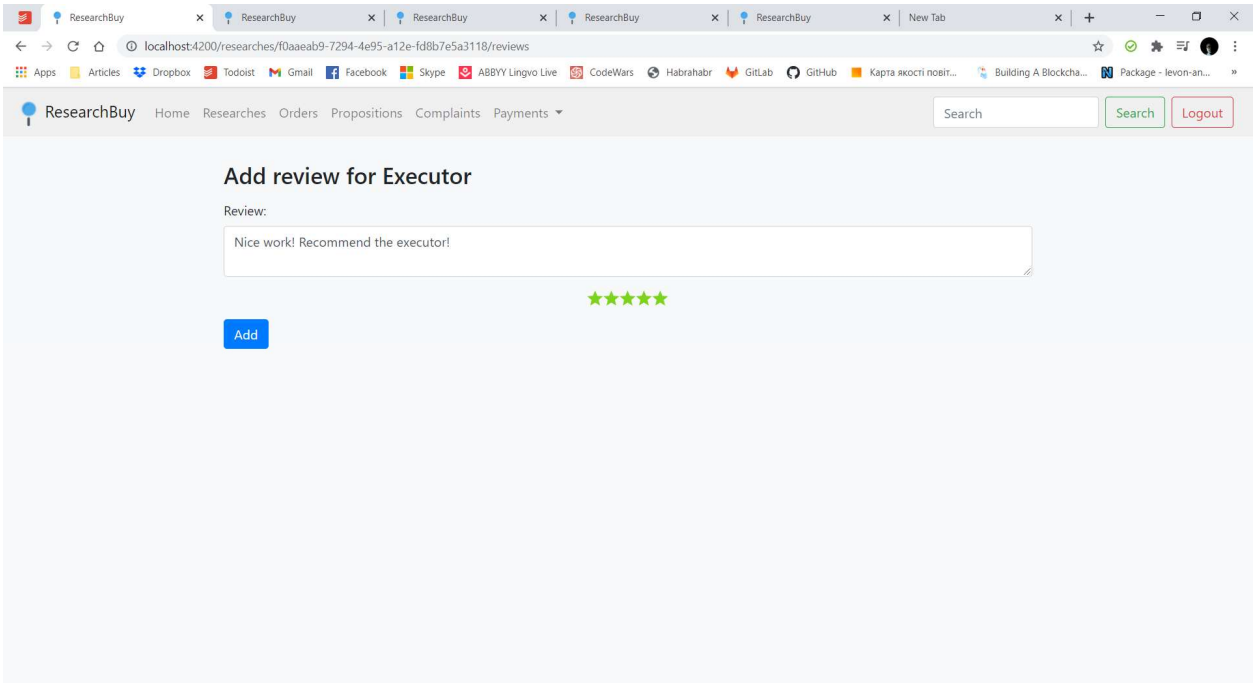


Рис. 4.12. Сторінка для відгуку про виконання дослідження

На рис. 4.12 зображена сторінка для створення відгуку про виконане дослідження. Власник завдання може залишити його після того, як виконавець завершить завдання. Для створення відгуку необхідно ввести його вміст та за допомогою зірочок виставити рейтинг виконавця, від 1 до 5. Після того, як користувач залишить відгук на сторінці профілю користувача відобразиться оновлений рейтинг та відгук про виконання завдання.

4.2.11. Огляд функціоналу адміністратора веб-сайту

Зробимо огляд функціоналу доступного адміністратору веб-сайту.

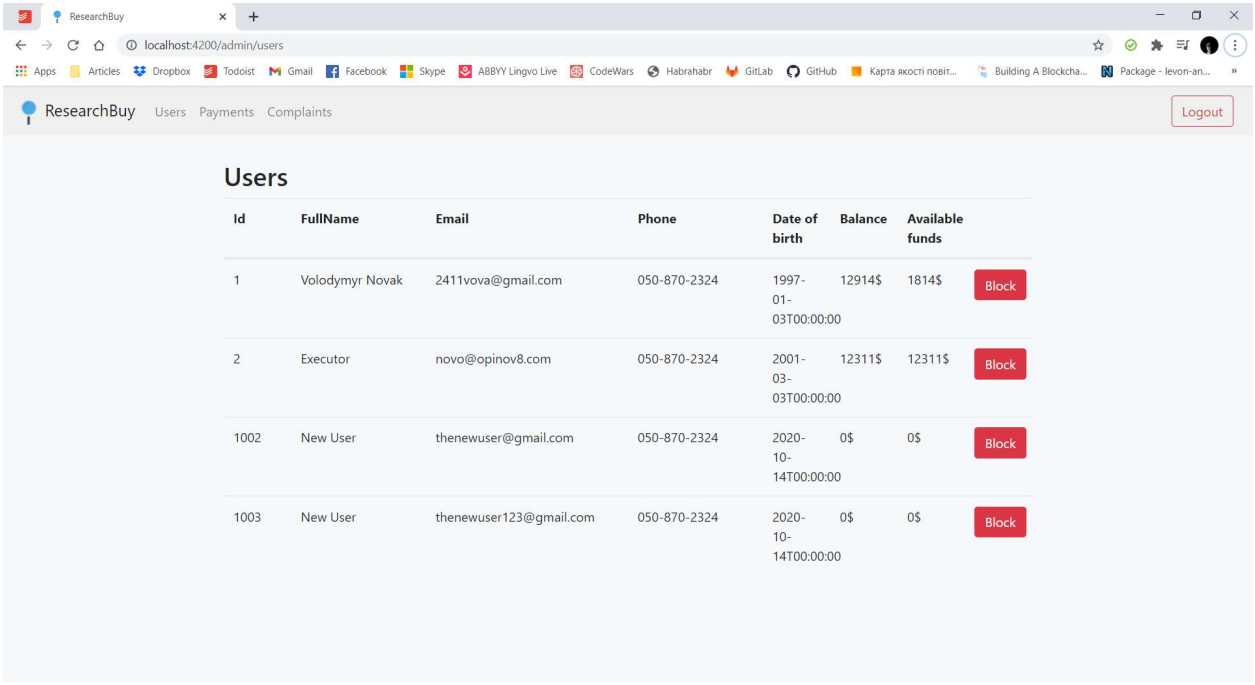


Рис. 4.13. Сторінка перегляду поточних користувачів в системі

На рис. 4.13 зображено список користувачів, які є у системі. При необхідності можна переглянути необхідні дані про них такі як:

- Ім’я
- Пошта
- Телефон
- Дата народження
- Поточний баланс
- Доступні кошти

Якщо користувач здійснює протиправні дії або порушує політики сайту – його можна заблокувати, натиснувши кнопку «Block». Якщо з якоїсь причини потрібно розблокувати користувача, то у рядку із заблокованим користувачем буде кнопка «Unblock».

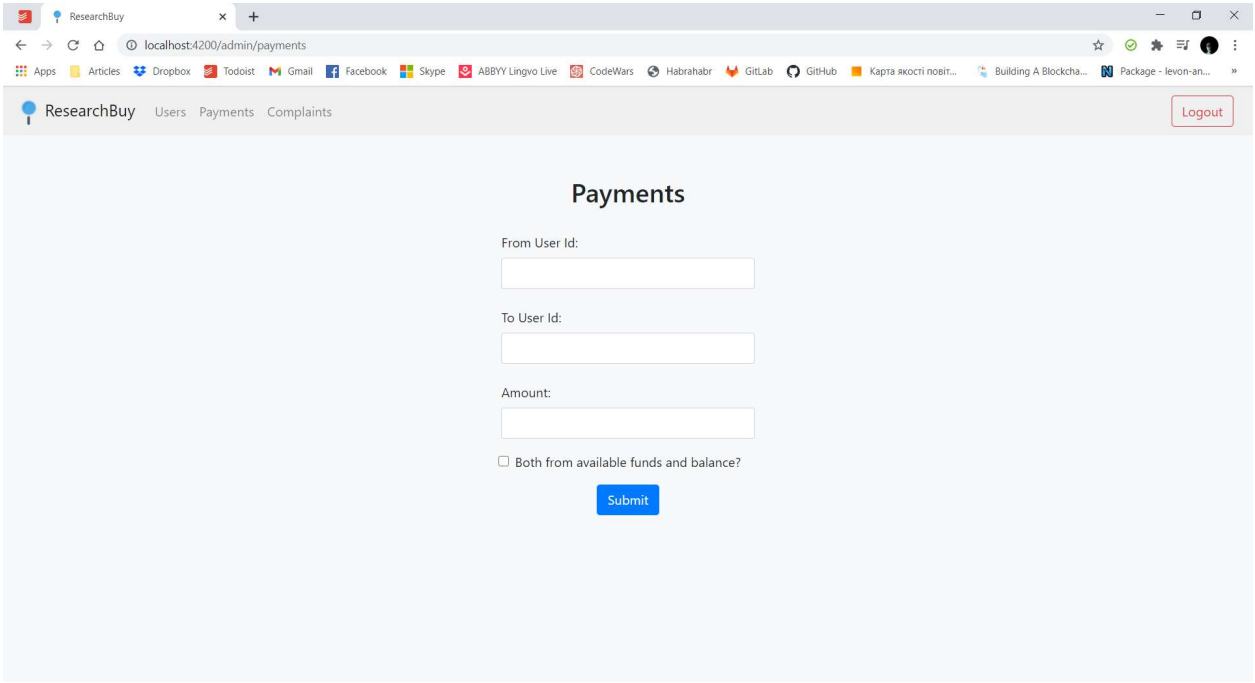


Рис. 4.14. Сторінка здійснення переказу коштів між гаманцями користувачів

При вирішенні спорів інколи виникає необхідність переказу коштів між гаманцями користувачів (рис. 4.14), якщо припустимо замовник виконання завдання з якихось причин не зміг здійснити оплату.

Для переказу потрібно ввести ідентифікатор користувача з рахунку якого будуть переводитись кошти, ідентифікатор користувача, на рахунок якого будуть зараховані кошти та суму операції. Ідентифікатори користувачів можна взяти з попереднього екрану зі списком користувачів. Він знаходиться у колонці «Id».

На екрані зображено також додаткове поле з галочкою. Якщо ми його відмітимо кошти будуть відняті як від балансу, так і від доступних зараз коштів, інакше вони будуть відняті тільки від балансу.

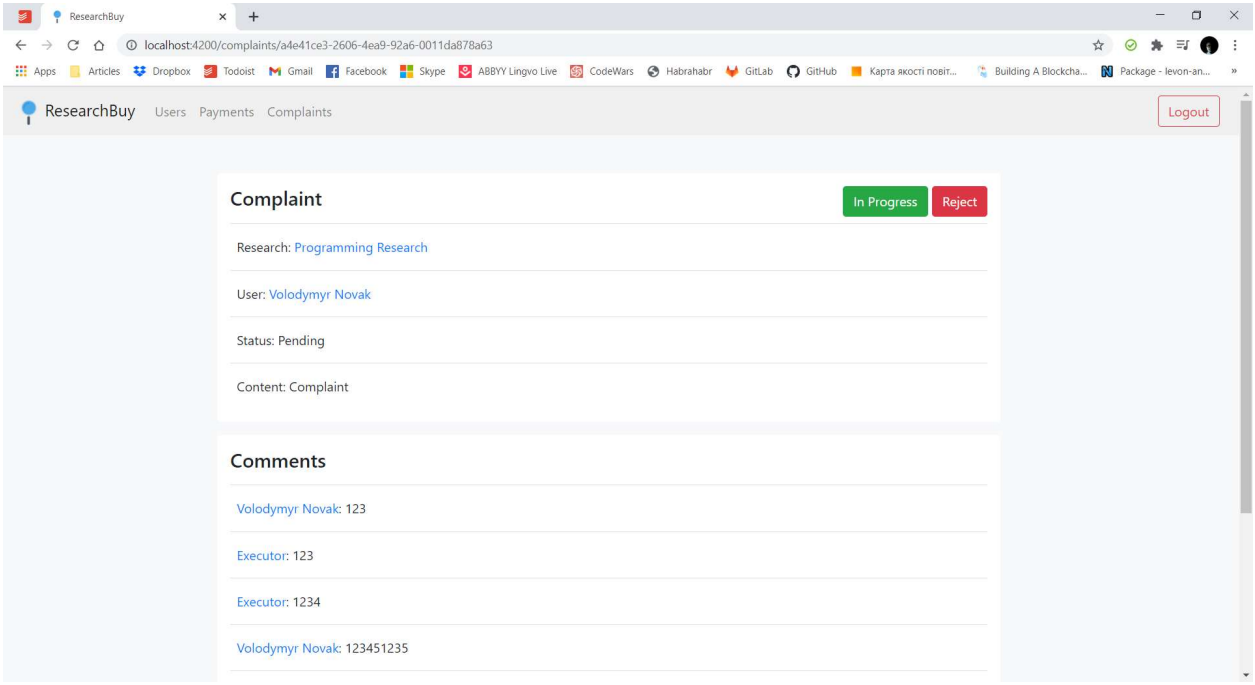


Рис 4.15. Сторінка з перегляду скарги

Після створення скарг користувачами, адміністратор має можливість їх переглядати та обробляти (рис. 4.15). Якщо скарга не є валідною, то адміністратор може її відхилити, натиснувши кнопку «Reject». При початку роботи з скаргою адміністратор веб-сайту має взяти її в роботу за допомогою кнопки «In Progress». Після закінчення роботи зі скаргою, якщо ініціатор її створення був правий і його вимоги були задоволені, адміністратор має натиснути на кнопку «Accept». Для комунікації з користувачем, який створив скаргу, адміністратор може використати систему коментування.

ВИСНОВКИ ДО РОЗДІЛУ 4

У розділі 4.1 було розглянуто опис реалізації предмету розробки магістерської дисертації. Було детально описано рішення, які були імплементовані на рівнях БД, серверного та клієнтського додатків.

У огляді структури бази даних було детально описано існуючі таблиці, їх вміст та зв'язки між ними. Для реалізації збереження бізнес-даних було використано реляційну БД SQL Server. Сутності було згенеровано за допомогою бібліотеки Entity Framework Core та підходу Code First.

У підрозділі про реалізацію серверного рівню додатку був проведений огляд всіх проектів, які існують у рішенні. Було детально розглянуто контролери та точки для взаємодії з системою, які є доступними для клієнтського додатку. Також було проведено детальний опис методів, які використовуються контролерами для виконання конкретної бізнес-логіки. Серверне рішення було реалізоване на платформі .NET Core з використанням бібліотек ASP.NET Core та Entity Framework Core.

Опис клієнтського рівню додатку містить огляд компонентів та їх призначень. Для розробки даного рівню було використано веб-фреймворк Angular і стандартні інструменти для розробки клієнтських додатків – HTML, CSS та JS.

У розділі 4.2 було розглянуто експлуатацію предмету розробки магістерської дисертації. Було детально оглянуто сторінки, які є доступними для користувача та описано дії, які може виконувати користувач на даних сторінках.

РОЗДІЛ 5.

РОЗРОБКА СТАРТАП ПРОЕКТУ

5.1. Опис ідеї проекту

На даний момент, бізнес розробки рішень для виконання досліджень та завдань не є новим. В розділі 1 було проведено детальний аналіз найбільш наближених аналогів до створеного рішення. Розроблене рішення є демонстрацією ідеї про те, що задачі, які призначені для певних виконавців можна ефективно та безпечно розділяти на частини, забезпечуючи поетапну перевірку результатів дослідження для власників, а для виконавців – швидше надходження коштів по мірі їх імплементації. У майбутньому дане рішення можна суттєво розширити, додавши функціонал, що дозволить краще конкурувати з наявними рішеннями на ринку.

Відповідно, в ході реалізації дипломної роботи було реалізовано ідею створення комплексного веб-рішення для забезпечення розробки системи захищеного та фрагментарного виконання досліджень. У цьому розділі буде описано розробку стартап-проекту даного рішення. В таблиці 5.1 буде описано вміст ідеї, що пропонується та вказано конкретні вигоди для користувачів.

Таблиця 5.1

Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Веб-рішення для забезпечення розробки системи захищеного та фрагментарного виконання досліджень	Створення досліджень	Користувач має можливість задокументувати деталі проведення дослідження, розбити його на частини, перенести його в цифрову форму
	Виконання досліджень	Виконавець має можливість зробити дослідження та отримати за нього винагороду, за виконання кожної з частин. При успішному завершенні дослідження, він може отримати схвальний відгук і в майбутньому отримати більше клієнтів. Власник завдання має вигоду в тому, що йому допомагають виконати задачу, має можливість переглядати результати виконання.
	Можливість пошуку досліджень до виконання	Користувач, який веде пошук роботи, може знайти дослідження, яке його цікавить та взяти його до виконання. Також є можливість переглянути, які дослідження є актуальними в даний час.

Зробимо аналіз потенційних техніко-економічних переваг ідеї та чим вона відрізняється від існуючих аналогів та замінників. Конкурентами даного рішення є такі ресурси як Freelancer.com, Upwork та marketing.rbc.ru.

Таблиця 5.2

Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№	Техніко-економічні характеристики ідеї	Мій проект	Конкуренти			W	N	S
			1	2	3			
1	Створення дослідження	+	+	+	+		+	
2	Оплата дослідження за виконану частину	+	+	+	-			+
3	Захист виконавця від випадку, в якому його робота залишиться неоплаченою (у разі успішного виконання завдання)	+	-	-	-			+
4	Можливість обговорення пропозиції по завданням	+	+	+	+		+	

5	Можливість переглянути результат виконання певної частини	+	+/-	+/-	-			+
6	Оцінка виконання завдання та відображення рейтингу користувача	+	+	+	-	+		
7	Наявність вбудованого месенджеру для комунікації користувачів	-	+	+	-	+		

Вказаний перелік слабких, сильних та нейтральних сторін ідеї даного продукту дає можливість йому конкурувати на ринку уже існуючих товарів. З табл. 5.2 слідує, що основними перевагами продукту є надання оплати за виконану частину завдання, захист як виконавця, так і власника завдання від шахраїв, можливість перегляду результатів виконання частини дослідження. На даний момент, недоліками додатку є те, що у ньому немає вбудованого месенджеру (вся комунікація відбувається в коментарях).

5.2. Технологічний аудит проекту

Таблиця 5.3

Технологічна здійсненність ідеї проекту

№	Ідея проекту	Технології її реалізації	Наявність технології	Доступність технологій
1	Веб-рішення для забезпечення розробки системи захищеного та фрагментарного виконання досліджень	Мова програмування C#	Є в наявності	Доступні безкоштовно
2		Платформа .NET Core	Є в наявності	Доступні безкоштовно
3		Бібліотека ASP.NET Core	Є в наявності	Доступні безкоштовно
4		Бібліотека Entity Framework Core	Є в наявності	Доступні безкоштовно
5		База даних SQL Server	Є в наявності	Доступні безкоштовно
6		Розподілене файлове сховище для даних IPFS	Є в наявності	Доступні безкоштовно
7		Веб-фреймворк Angular	Є в наявності	Доступні безкоштовно

У ході планування розробки предмету дослідження магістерської дисертації було вирішено розробити клієнт-серверну систему. Для розробки серверної частини використано технології .NET Core, ASP.NET Core, EF Core, SQL Server та IPFS. Більшість коду написана на мові програмування C#. Клієнтська частина була розроблена на базі веб-фреймворку Angular.

5.3. Аналіз ринкових можливостей запуску стартап проекту

Проведемо аналіз ринкових можливостей запуску стартап проекту. В таблиці 5.4. наведено попередню характеристику потенційного ринку стартап-проекту.

Таблиця 5.4

Попередня характеристика потенційного ринку стартап-проекту

№	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	>10
2	Динаміка ринку (якісна оцінка)	Стагнує
3	Наявність обмежень для входу (вказати характер обмежень)	Конкуренція для входу на ринок, необхідність напрацювання бази як і виконавців, так і людей, які будуть готові створювати досліджування
4	Специфічні вимоги до стандартизації та сертифікації	-

Таблиця 5.5

Характеристика потенційних клієнтів стартап-проекту

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Декомпозиція виконання дослідження на частини	Власники досліджень, які зацікавлені в поетапному та чіткому виконанню вимог.	Різні потенційні цільові групи клієнтів можуть надавати різні вимоги до етапів виконання дослідження, це може бути вказано в його описі	Можливість ефективно та зручно розділяти дослідження на частини, відслідковувати прогрес по кожній з них
2	Створення можливості роздільної оплати за різні частини дослідження	Виконавці, які хочуть швидше отримувати оплату за виконання роботи	Виконавці зацікавлені у своєчасному та швидкому надходженню коштів. Разом з тим, власники досліджень зацікавлені у якісному виконанні завдань.	Для початку роботи над частиною дослідження, замовник має мати кошти на рахунку для її оплати. У разі недобросовісного виконання кошти будуть повернені власнику. Інакше – кошти переходять на рахунок виконавця.

3	Захист виконавця від випадку, в якому його робота залишиться неоплаченою	Виконавці, які мають негативний досвід з клієнтами, які не виплатили винагороду	Виконавці зацікавлені у оплаті виконаної роботи	Створення механізму виплати винагород, при якому не може бути випадку, коли виконавець залишиться без оплати

Таблиця 5.6

Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренція на ринку	На даний момент, на ринку є досить зрілі рішення, які мають велику кількість користувачів та багатий функціонал	Розширення існуючого функціоналу додатку, створення умов, при яких користувачам буде зручніше та вигідніше працювати саме з нашим рішенням
2	Маркетинговий	Просування товару на ранньому етапі є надзвичайно важливим для життєздатності продукту. Якщо про продукт ніхто не дізнається, то у нас не буде користувачів.	Необхідно знайти команду спеціалістів, яка зможе ефективно залучити користувачів за допомогою реклами, рекомендації і так далі
3	Ресурсний	Для того, щоб конкурувати з провідними рішеннями на ринку потрібна велика кількість передусім грошових	Потрібно залучити інвесторів. Можливо, вийти на відкриті майданчики для залучення інвестицій. Відповідно,

		ресурсів для найму додаткових спеціалістів по програмному забезпеченню, маркетологів. Потрібно продумати структуру та ціну хостингу.	для меценатів, які пожертвували гроші на ранньому етапі, у майбутньому надавати певні преференції при використанні продукту.
4	Розширення спектру фунціоналу	Необхідно провести ширші дослідження серед цільової аудиторії і доповнити додаток функціоналом, який є критичним для них і відсутній, на даний момент, в додатку.	Проведення широкого дослідження серед потенційних користувачів системи.

Таблиця 5.7

Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Приваблення інвестицій	Якщо компанії вдасться створити бізнес-план, який задовільнить інтерес інвесторі, ми отримаємо суттєві додаткові грошові ресурси	Залучення додаткових людей у штат, пришвидшення розробки додатку
2	Монетизація розробленого рішення	Додавання реклами в додаток, різних типів користувацьких профілів (введення платних підписок для розширеного функціоналу)	Збільшення прибутків від розробленого рішення.
3	Розширення кола користувачів додатку	Після проведення маркетингової компанії, коло користувачів розшириться	При розширенні додаткового кола користувачів є можливість залучення додаткових інвестицій, які можна витратити на покращення продукту та удосконалення інфраструктури для підтримки більшої кількості користувачів.
4	Створення нового рішення у суміжній галузі	При вдалому старті продукту та припливу інвестицій, можна задуматись над розробкою нового продукту у суміжній області, який буде мати приблизно схожу цільову аудиторію	Залучення інвестицій, розширення штабу, проведення опитувань серед цільових користувачів.

Таблиця 5.8

Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентос-проможною)
Тип конкуренції: чиста	Конкуренція є чистою, так як у кожного з великих гравців на ринку є своя аудиторія користувачів, звичайно, є лідери ринку, проте, це важко назвати монополією або олігополією	Випуск товару, який має підтримувати найбільш важливий та цікавий, який реалізований у лідерів ринку, додавши до нього власні унікальні покращення.
За рівнем конкурентної боротьби: глобальна	Лідери на ринку представлені у багатьох країнах світу	Підтримка різних мов у додатку, підтримка можливості оплати через провідні світові платіжні системи.
За галузевою ознакою: внутрішньо-галузева	Загалом, товари конкурують переважно всередині однієї галузі.	Потрібно забезпечити ефективні конкурентні недоліки в порівнянні з аналогами у межах однієї галузі.
За видами товарів: товарно-видова	Конкуренція відбувається між товарами одного виду	Потрібно зробити товар одним з лідерів у своєму виді.
За характером конкурентних переваг: цінова і нецінова	У даному випадку, не можна вибрати конкретного рішення, так як, для користувачів є важливим як фактор ціновий – комісії у платіжних системах, відсоток від компанії, так і неціновий – зручність та продуктивність системи	Потрібно працювати у двох напрямках паралельно. Зменшувати комісії на переводи коштів. Удосконалювати зручність та якість додатку.
За інтенсивністю: марочна	На ринку зараз є більше десятки компаній, які пропонують схожі товари та послуги, при цьому кожен з	Потрібно нав’язати конкурентам достойну альтернативу, яка буде мати яскраво виражені

	них хоче стати лідером та монополістом на ринку.	сильні сторони.

Таблиця 5.9

Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти в галузі	Постачальники	Клієнти	Товари-замінники
	Upwork, Freelancer.com, marketing.rbc.ru	Researchgate, Toptal	Продуктові компанії, які розробили конкурентні продукти	Замовники та виконавці досліджень	Upwork, Freelancer.com
Висновки:	На ринку уже існують готові та працюючі рішення з великою базою клієнтів. Конкуренція з їхнього боку є сильною.	У галузі також є потенційні конкуренти, які виконують схожі функції, проте, менше зав’язані власне на вирішення досліджень	Зазвичай, постачальниками послуг є продуктові компанії, які розробили дані рішення.	Потенційні клієнти розробленого рішення уже використовують наявні на ринку рішення.	Конкуренція буде нав’язана за допомогою покращення існуючих підходів, підвищення ефективності роботи та доходів користувачів.

Таблиця 5.10

Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Забезпечення декомпозиції досліджень на частини	Вбудоване розділення дослідження на частини. Можливість для виконавців вводити корекцію етапів виконання та пропонування власних цін для кожної з частин. Перегляд виконаних частин зі сторони замовника та надання відгуку про їх якість.
2	Роздільна оплата при виконанні дослідження	Перед початком виконання певної частини, замовник має внести кошти для її виконання. При неякісному виконанні дослідження замовник може залишити скаргу і кошти будуть розблокованими. Якщо ж все буде зроблено за вимогами, виконавець зможе швидше отримати оплату.
3	Захищене збереження результатів виконання дослідження	В порівнянні з конкурентами, в розробленій системі одним з основних завдань було поставлено покращити збереження результатів виконання досліджень за допомогою IPFS та власної реалізації алгоритму Blockchain.

Таблиця 5.11

Порівняльний аналіз сильних та слабких сторін

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з розроблюваним продуктом						
			-3	-2	-1	0	1	2	3
1	Забезпечення декомпозиції досліджень на частини	10				✓			
2	Роздільна оплата при виконанні дослідження	14		✓					
3	Захищене збереження результатів виконання дослідження	15		✓					

Таблиця 5.12

SWOT- аналіз стартап-проекту

Сильні сторони: Забезпечення декомпозиції досліджень на частини Роздільна оплата при виконанні дослідження Захищене збереження результатів виконання дослідження Покращення у новому продукті слабких сторін у прямих конкурентів	**Слабкі сторони:** Висока конкуренція Наявність готових рішень Необхідність залучення клієнтів з конкуруючих платформ
Можливості: Залучення інвестицій Монетизація рішення за рахунок реклами та платних підписок Створення нових рішень у суміжних галузях на основі існуючої клієнтської бази	**Загрози:** Відсутність інвестицій Програш конкуренції Невелика кількість користувачів при програшній маркетинговій компанії

Таблиця 5.13

Альтернативи ринкового впровадження стартап-проекту

№	Альтернатива ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Переробка проекту під некомерційне середовище для збереження важливих досліджень у державних установах	Середня, так як зараз в нашій державі йдуть позитивні тенденції, в плані, впровадження технічних новинок в державний апарат. Проте, все ж існують проблеми з прозорістю та корупцією при державних закупівлях	До року, при необхідності можна доробити функціонал додатку відповідно до нових вимог
2	Розповсюдження проекту через opensource	Низька, проект можуть використати в своїх цілях інші люди без відома розробників.	Кілька днів
3	Залучення інвестицій до проекту через відкриті майданчики, такі як Kickstarter	Висока, можна отримати інвестиції від зацікавлених потенційних користувачів та дізнатись більше про їхні потреби	Рік та більше, в залежності від зацікавленості аудиторію та об’єму інвестицій

5.4. Розроблення ринкової стратегії проекту

Таблиця 5.14

Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Індивідуальні замовники досліджень	Не всі готові сприйняти негайно	Продукт необхідний	Конкуренція висока	Складний вхід
2	Приватні компанії	Зазвичай, для досліджень використовують великі компанії, які на цьому спеціалізуються	Продукт необхідний не всім	Конкуренція висока	Складний вхід
3	Виконавці досліджень (науковці, індивідуальні підприємці, студенти)	Готові прийняти продукт	Продукт необхідний (покриває деякі важливі недоліки в існуючих конкурентів)	Конкуренція висока	Складний вхід
Які цільові групи обрано: Потрібно сконцентруватись на залученні індивідуальних замовників досліджень та виконавцях.					

Таблиця 5.15

Визначення базової стратегії розвитку

№	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентос-проможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Розвиток проекту шляхом демонстрації переваг проекту для потенційних інвесторів	Необхідно максимально сконцентруватись на вимогах цільової аудиторії і переманити на свою сторону малих та середніх замовників.	При проходженні даного шляху продукт має більший шанс на успіх через залучення більшої кількості ресурсів та розширення кількості ідей за рахунок пропозицій інвесторів.	Прислухатись до побажань кінцевих користувачів та інвесторів. Підвищувати якість та функціонал додатку.

Таблиця 5.16

Визначення базової стратегії конкурентної поведінки

№	Чи є проект «пер-шопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
1	Ні	Переважно, компанія буде займатись переманюванням існуючих користувачів у конкурентів. Паралельно, буде здійснений пошук нових користувачів.	Так, компанія буде копіювати основні переваги у конкурентів, такі як робота у різних країнах світу, підтримка різних типів робіт.	Забезпечити більш зручний та ефективний спосіб роботи, ніж у конкурентів. Перебрати всі основні переваги і усунути недоліки.

Таблиця 5.17

Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкуренто-спроможні позиції власного стартап- проекту	Вибір асоціацій, які мають сфо-рмувати комплексну позицію власного проекту (три ключових)
1	Декомпозиція дослідження на частини, роздільна оплата досліджень, пришвидшення виводу коштів	Розширення функціоналу додатку. Збільшення персоналу. Фокус на якість та безпеку.	Швидка реакція на потреби користувачів. Впровадження нових функцій.	Система для виконання досліджень, робота з дослідженнями.

5.5. Розробка маркетингової програми стартап-проекту

Таблиця 5.18

Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	Забезпечення декомпозиції досліджень на частини	Ефективний механізм для роботи з дослідженням по частинам, можливість перегляду роботи на різних етапах.	Розбивання дослідження на частини, можливість пропонування виконавцями своїх частин перед виконанням дослідження
2	Роздільна оплата при виконанні дослідження	Впровадження механізму оплати при виконанні частини дослідження	На відміну, від конкурентів виконавці можуть швидше отримувати кошти.
3	Захищене збереження результатів виконання дослідження	В порівнянні з конкурентами збільшено фокус на захищене збереження результатів досліджень	Зменшення ймовірності крадіжки результатів досліджень.

Таблиця 5.19

Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="2">Сутність та складові</td></tr>
<tr><td>I. Товар за задумом</td><td colspan="2">Система для захищеного та фрагментарного виконання досліджень, для забезпечення ефективної та продуктивної взаємодії виконавців та замовників.</td></tr>
<tr><td rowspan="5">II. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>Розмір</td></tr>
<tr><td>1. Клієнтський додаток на базі фреймворку Angular</td><td>352 MB</td></tr>
<tr><td>2. Серверний додаток на базі плафторми .NET Core</td><td>217 MB</td></tr>
<tr><td colspan="2">Якість: тестування програми на середовищі, логування виключень, перевірка продуктивності роботи додатку.</td></tr>
<tr><td colspan="2">Пакування: код продукту не передбачений для продажу. Доступ до нього буде вільним через мережу Інтернет.</td></tr>
<tr><td></td><td colspan="2">Марка: логотип та назва товару.</td></tr>
<tr><td rowspan="2">III. Товар із підкріпленням</td><td colspan="2">До продажу: програмний продукт не призначений для продажу.</td></tr>
<tr><td colspan="2">Після продажу: програмний продукт не призначений для продажу.</td></tr>
<tr><td colspan="3">За рахунок чого потенційний товар буде захищено від копіювання: код програмного продукту захищено від копіювання через збереження його коду у приватному репозиторії, вихідні артефакти коду на серверній частині планується захистити за допомогою механізму обфускації.</td></tr>
</table>

Таблиця 5.20

Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі становлення ціни на товар/послугу
1	0-60$	0-100$	>200$	0-30$

Таблиця 5.21

Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових кліє-нтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Доступ до сайту через мережу Інтернет. Прямі аналоги надають можливість використовувати безкоштовні акаунти.	Хостинг сайту на одному з популярних провайдерів.	Однорівневий	Вертикальна

Таблиця 5.22

Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Через онлайн-ресурси та рекламу дізнаються про продукт	Інтернет, зустрічі з колегами, перегляд конференцій.	Система для захищеного та фрагментарного виконання досліджень	Привернути увагу потенційних користувачів до додатку, показати їм основні переваги, залучити до використання платформи	Показати потенційному користувачу ситуацію, коли він не задоволений існуючими продуктами та продемонструвати переваги розробленого рішення

ВИСНОВКИ ДО РОЗДІЛУ 5

У п’ятому розділі магістерської дисертації було описано розробку стартап-проекту на базі створеного програмного рішення. В результаті виконання розділу було здійснено:

- Аналіз ідеї проекту, можливих напрямків застосування проекти, виділено основні вигоди, які може отримати користувач від товару.
- Опис технологічного аудиту, перераховано список технологій, які були використані для розробки рішення.
- Визначення ринкових можливостей запуску стартап-проекту, які можна використати при впровадженні проекту, та загроз, що існують на ринку.
- Розроблено ринкову стратегію проекту, проведено опис основних цільових груп потенційних користувачів, визначено базову стратегію розвитку.
- Створено маркетингову програму проекту для реалізації продукту.

Отже, було створено стартап-проект для запуску імплементованого програмного рішення на ринок, набуто цінні навички з запуску та створення стартап-проектів, дослідження та аналізу цільового ринку продукту.

ВИСНОВКИ

Система для захищеного та фрагментарного виконання досліджень може удосконалити та допомогти людям, які працюють з дослідженнями. У ході виконання роботи була досягнена її мета - забезпечення ефективного, зручного та захищеного механізму продажу досліджень, з можливістю декомпозиції завдання на частини.

Для досягнення мети було розроблено комплексний клієнт-серверний веб-додаток, який складається з 3 рівнів: клієнтський, серверний та БД. Було проведено комплексний аналіз технологій, які можна використати для створення рішення. На етапі планування було вирішено використати для клієнтського додатку веб-фреймворк Angular, для серверного додатку – платформу .NET Core та бібліотеку ASP.NET Core. Як основну БД, було обрано SQL Server. Для збереження результатів досліджень було використано розподілене файлове сховище IPFS.

У результаті було реалізовано програмний продукт, який надає можливості для створення досліджень, розбиття їх на частини, пропозиції власного плану виконання досліджень. Також був імплементований механізм роздільної оплати за різні частини дослідження. Було здійснено фокус на захищене збереження результатів виконання досліджень за допомогою IPFS у зв’язці з базовою імплементацією алгоритму Blockchain. Після розробки продукту було проведене мануальне тестування рішення, усунуто недоліки непомічені на стадії створення програмного коду.

Паралельно з розробкою програмної частини магістерської дисертації була проведена розробка стартап-проекту. В результаті дослідження було визначено, що основними перевагами над прямими конкурентами є забезпечення декомпозиції досліджень на частини, роздільна оплата при виконанні дослідження та захищене збереження результатів виконання дослідження. Також було виявлено наступні недоліки при вході на ринок: висока конкуренція, наявність готових рішень та необхідність залучення

клієнтів з конкуруючих платформ. При підготовці стартап-проекту було пропрацьовано ринкову та маркетингову стратегії.

СПИСОК ВИКОРИСТАНОЇ ДЖЕРЕЛ

1. The great transformer: The impact of the Internet on economic growth and prosperity [Електронний ресурс] - Режим доступу до ресурсу: https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/the-great-transformer. – Дата перегляду: 10.09.2020.
2. What Are the Benefits of the Internet to Business? [Електронний ресурс] - Режим доступу до ресурсу: https://smallbusiness.chron.com/benefits-internet-business-316.html. – Дата перегляду: 10.09.2020.
3. Review of Freelancer.com [Електронний ресурс] - Режим доступу до ресурсу: https://blog.hubstaff.com/review-of-freelancer-com/. – Дата перегляду: 17.09.2020.
4. Upwork Reviews: Is It Worth It? [Електронний ресурс] - Режим доступу до ресурсу: https://biz30.timedoctor.com/upwork-review/. – Дата перегляду: 17.09.2020.
5. РБК. Исследования рынков [Електронний ресурс] - Режим доступу до ресурсу: https://www.retail.ru/rbc/company/rbk_issledovaniya_rynkov/. – Дата перегляду: 18.09.2020.
6. Клиент-серверная архитектура в картинках [Електронний ресурс] - Режим доступу до ресурсу: https://habr.com/ru/post/495698/. – Дата перегляду: 21.09.2020.
7. Angular Single Page Applications (SPA): What are the Benefits? [Електронний ресурс] - Режим доступу до ресурсу: https://blog.angular-university.io/why-a-single-page-application-what-are-the-benefits-what-is-a-spa/. – Дата перегляду: 22.09.2020.
8. Файн, Я. Angular и TypeScript. Сайтостроение для профессионалов / Я. Файн. – С.Пб. : Питер, 2013. - 464 с

9. Dependency injection in Angular [Електронний ресурс] - Режим доступу до ресурсу: https://angular.io/guide/dependency-injection. – Дата перегляду: 22.09.2020.
10. TypeScript for JavaScript Programmers [Електронний ресурс] - Режим доступу до ресурсу: https://www.typescriptlang.org/docs/handbook/typescript-in-5-minutes.html. – Дата перегляду: 22.09.2020.
11. Редактор кода Visual Studio Code. Самый подробный гайд по настройке и установке плагинов для начинающих [Електронний ресурс] - Режим доступу до ресурсу: https://habr.com/ru/post/490754/. – Дата перегляду: 23.09.2020.
12. The ASP.NET Core Revolution – .NET Core history through the years (2016-2019) [Електронний ресурс] - Режим доступу до ресурсу: https://smartworknet.eu/the-asp-net-core-revolution-net-core-history-through-the-years-2016-2019/. – Дата перегляду: 30.09.2020.
13. Create web APIs with ASP.NET Core [Електронний ресурс] - Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/aspnet/core/web-api/?view=aspnetcore-5.0. – Дата перегляду: 30.09.2020.
14. SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST / T.Erl, B. Carlyle, C. Pautasso, C. Balasubramanian., 2012. – 624 с.
15. Шилдт Герберт. C# 4.0: Полное руководство / Герберт Шилдт. - Вильямс, 2011. - 1056 с
16. Рихтер Джеффри. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. / Джеффри Рихтер. - Питер, 2013. - 896 с.
17. SQL vs NoSQL: 5 Critical Differences [Електронний ресурс] - Режим доступу до ресурсу: https://www.xplenty.com/blog/the-sql-vs-nosql-difference/. – Дата перегляду: 10.10.2020.

18. Codd's Rules for RDBMS [Електронний ресурс] - Режим доступу до ресурсу: https://www.w3resource.com/sql/sql-basic/codd-12-rule-relation.php
19. Lewis P. M. Database and Transaction Processing / P. M. Lewis, A. Bernstein, M. Kifer., 2001. – 1056 с.
20. JSON Web Token (JWT) [Електронний ресурс] - Режим доступу до ресурсу: https://tools.ietf.org/html/rfc7519. – Дата перегляду: .
21. Introduction to authorization in ASP.NET Core [Електронний ресурс] - Режим доступу до ресурсу: https://docs.microsoft.com/ru-ru/aspnet/core/security/authorization/introduction?view=aspnetcore-5.0. – Дата перегляду: 13.10.2020.
22. What is IPFS? [Електронний ресурс] - Режим доступу до ресурсу: https://docs.ipfs.io/concepts/what-is-ipfs/. – Дата перегляду: 14.10.2020.
23. Interplanetary Hash Tables [Електронний ресурс] - Режим доступу до ресурсу: https://miro.medium.com/proxy/1*nSQLpYbXITdbnbiX7lZuig.png. – Дата перегляду: 14.10.2020.
24. Libp2p [Електронний ресурс] - Режим доступу до ресурсу: https://docs.ipfs.io/concepts/libp2p/. – Дата перегляду: 15.10.2020.
25. What is Blockchain? [Електронний ресурс] - Режим доступу до ресурсу: https://www.guru99.com/blockchain-tutorial.html. – Дата перегляду: 20.10.2020.
26. Introduction to Vault [Електронний ресурс] - Режим доступу до ресурсу: https://www.vaultproject.io/docs/what-is-vault. – Дата перегляду: 23.10.2020.
27. Introduction to Code Obfuscation [Електронний ресурс] - Режим доступу до ресурсу: https://medium.com/better-programming/code-obfuscation-introduction-to-code-obfuscation-part-1-93a6797349b0. – Дата перегляду: 24.10.2020.
28. Obfuscar Documentation [Електронний ресурс] - Режим доступу до ресурсу: https://docs.obfuscar.com/. – Дата перегляду: 25.10.2020.
29. Smith J. P. Entity Framework Core in Action / Jon Smith., 2018. – 520 с.

30. Angular - HttpInterceptor [Електронний ресурс] - Режим доступу до ресурсу: https://angular.io/api/common/http/HttpInterceptor. – Дата перегляду: 27.10.2020.

ДОДАТКИ

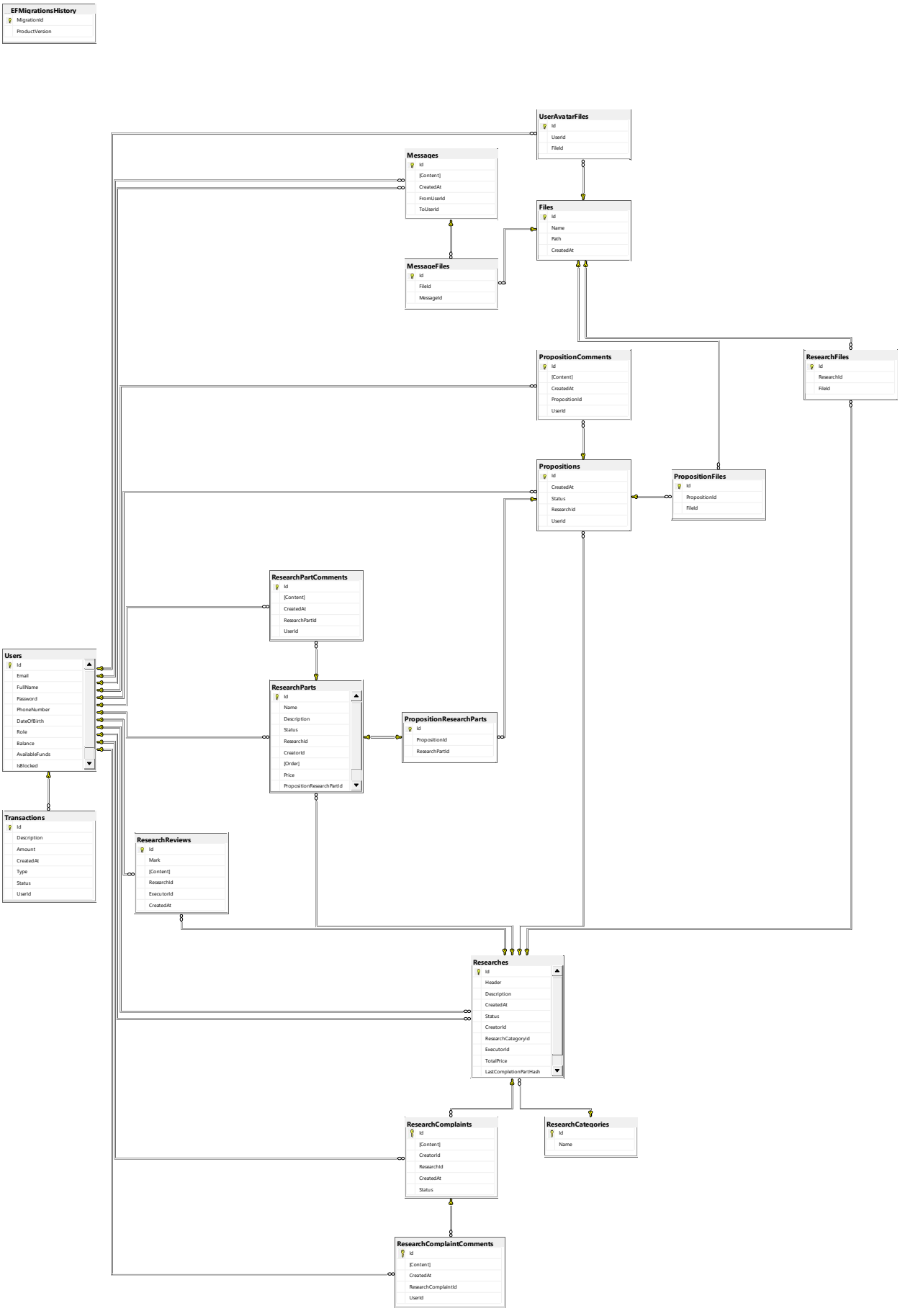
ДОДАТОК 1

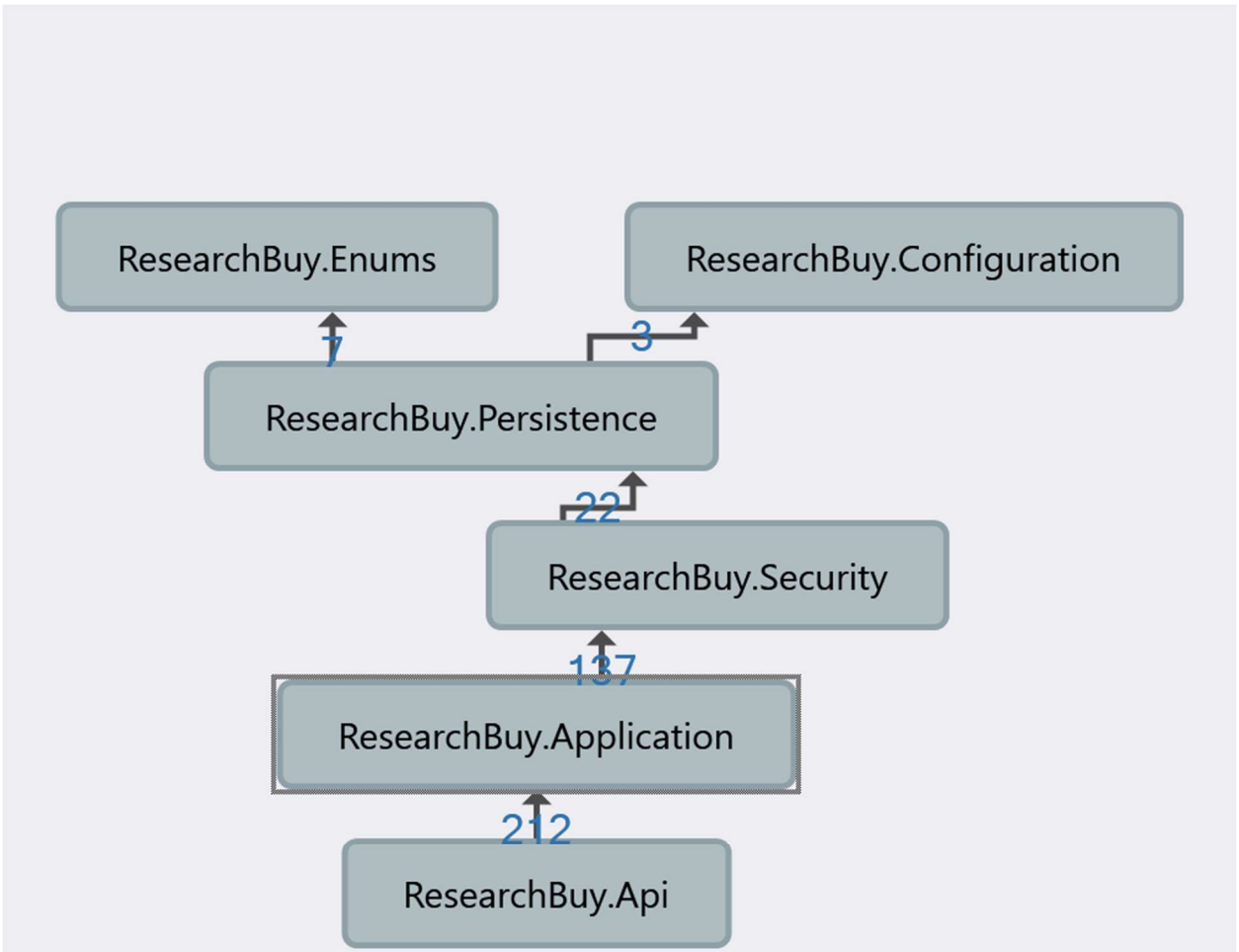
СИСТЕМА ДЛЯ ЗАХИЩЕНОГО ТА ФРАГМЕНТАРНОГО ВИКОНАННЯ ДОСЛІДЖЕНЬ

Діаграми

Аркушів 2

Київ - 2020


EFMigrationsHistory
MigrationId
ProductVersion
UserAvatarFiles
Id
UserId
FileId
Messages
Id
[Content]
CreatedAt
FromUserId
ToUserId
Files
Id
Name
Path
CreatedAt
MessageFiles
Id
FileId
MessageId
PropositionComments
Id
[Content]
CreatedAt
PropositionId
UserId
ResearchFiles
Id
ResearchId
FileId
Propositions
Id
CreatedAt
Status
ResearchId
UserId
PropositionFiles
Id
PropositionId
FileId
ResearchPartComments
Id
[Content]
CreatedAt
ResearchPartId
UserId
Users
Id
Email
FullName
Pasword
PhoneNumber
DateOfBirth
Role
Balance
AvailableFunds
IsBlocked
ResearchParts
Id
Name
Description
Status
ResearchId
CreatorId
[Order]
Price
PropositionResearchPartId
PropositionResearchParts
Id
PropositionId
ResearchPartId
Transactions
Id
Description
Amount
CreatedAt
Type
Status
UserId
ResearchReviews
Id
Mark
[Content]
ResearchId
ExecutorId
CreatedAt
Researches
Id
Header
Description
CreatedAt
Status
CreatorId
ResearchCategoryId
ExecutorId
TotalPrice
LastCompletionPartHash
ResearchComplaints
Id
[Content]
CreatorId
ResearchId
CreatedAt
Status
ResearchCategories
Id
Name
ResearchComplaintComments
Id
[Content]
CreatedAt
ResearchComplaintId
UserId


ResearchBuy.Enums
ResearchBuy.Configuration
7
3
ResearchBuy.Persistence
22
ResearchBuy.Security
137
ResearchBuy.Application
212
ResearchBuy.Api

ДОДАТОК 2

СИСТЕМА ДЛЯ ЗАХИЩЕНОГО ТА ФРАГМЕНТАРНОГО ВИКОНАННЯ ДОСЛІДЖЕНЬ

Таблиці

Аркушів 11

Київ - 2020

Таблиця 2.1

Сервіси рівню додатку

Назва сервісу	Назва методу	Параметри	Опис методу
FileService	AddRange	Колекція файлів	Збереження колекції файлів у локальну файлову систему
	GetFileById	Ідентифікатор файлу	Зчитування вмісту файлу
	Create	Файл	Створення файлу і запис в локальну файлову систему
PaymentService	GetBalance	-	Зчитування балансу поточного користувача
	Deposit	Сума транзакції	Внесення коштів на рахунок користувача
	Withdraw	Сума транзакції	Виведення коштів з рахунку користувача
	Transfer	Сума транзакції, отримувач та відправник коштів	Здійснення переводу коштів між рахунками користувачів
	BlockFunds	Сума транзакції та користувач	Блокування суми на рахунку користувача

	UnblockFunds	Сума транзакції та користувач	Розблокування коштів на рахунку користувача
	ReserveFunds	Ідентифікатор частини дослідження	Проведення блокування коштів на рахунку користувача для виконання певної частини дослідження
PropositionService	Add	Ідентифікатор дослідження, запропоновані частини, ідентифікатори прикріплених файлів	Створення пропозиції для виконання певного дослідження
	GetById	Ідентифікатор пропозиції	Зчитування пропозиції для виконання певної пропозиції
	GetPropositions	-	Зчитування всіх пропозиції створених поточним користувачем
	Accept	Ідентифікатор пропозиції	Прийняття пропозиції користувачем, відповідно людина,

			яка створила пропозицію стає виконавцем завдання
	AddComment	Ідентифікатор пропозиції, вміст коментарю	Додання коментарю до пропозиції. Виклик бібліотеки SignalR для оновлення сторінки перегляду пропозиції новим коментарем
ResearchCategory Service	GetAll	-	Зчитування всіх існуючих категорій досліджень
ResearchComplaint Service	Add	Ідентифікатор дослідження, вміст скарги	Створення скарги при виконанні дослідження.
	GetAll Complaints	-	Зчитування усіх існуючих скарг в системі
	GetMy Complaints	-	Зчитування скарг, створених виключно поточним користувачем
	GetById	Ідентифікатор скарги	Зчитування скарги за ідентифікатором

	AddComment	Ідентифікатор скарги	Додавання коментарю до скарги.
	UpdateStatus	Ідентифікатор та статус скарги	Оновлення статусу скарги. Зазвичай, відбувається після її розгляду адміністратором сайту.
ResearchPart CommentService	AddComment	Ідентифікатор частини дослідження, вміст коментарю	Додавання коментарю до частини дослідження.
	GetComments	Ідентифікатор частини дослідження	Зчитування усіх коментарів по даній частині дослідження
ResearchPart Service	GetById	Ідентифікатор частини дослідження	Зчитування частини дослідження за вказаним ідентифікатором
	SubmitPart Execution Result	Ідентифікатор частини дослідження, опис результату,	Завантаження та збереження результатів виконання дослідження.

		прикріплені файли	
	GetFileContent	Ідентифікатор частини дослідження, ідентифікатор результату виконання, ідентифікатор файлу	Зчитування файлу, який міститься у результатах виконання дослідження.
	AcceptResultOfPart	Ідентифікатор частини дослідження та результату виконання дослідження	Прийняття певного результату виконання дослідження, як такого, який задовольняє всі умови власника завдання. Закриття частини завдання, як виконаної
	UpdateStatus	Ідентифікатор частини дослідження та статус	Оновлення статусу частини завдання
ResearchService	Add	Категорія дослідження, заголовок,	Створення нової частини дослідження,

		опис, прикріплені файли, частини	відповідно до вказаних параметрів
	GetMy Researches	-	Зчитування досліджень створених поточним користувачем
	GetMy Orders	-	Зчитування досліджень у яких поточний користувач є виконавцем
	FindResearches	Текст, ціна дослідження, категорія	Пошук досліджень, які ще не мають виконавців.
	GetResearch ById	Ідентифікатор дослідження	Зчитування дослідження за ідентифікатором
	AddResearch Review	Ідентифікатор дослідження, оцінка (1-5), відгук	Створення відгуку про виконання дослідження.
	GetReviews	-	Зчитування відгуків по поточному користувачу
	GetStatus	Ідентифікатор дослідження	Зчитування статусу дослідження за

			вказаним ідентифікатором
	Fail	Ідентифікатор дослідження	Дострокове закінчення дослідження через незадовільні результати по виконанню
UserService	Register	Електронна пошта, ім’я, дата народження, пароль, номер телефону	Створення нового користувача в системі
	IsUniqueEmail	Електронна пошта	Перевірка унікальності електронної пошти на базі існуючих користувачів
	SignIn	Електронна пошта та пароль	Вхід в систему, проведення аутентифікації користувача
	GetProfile	Ідентифікатор користувача	Зчитування профілю користувача за

			певним ідентифікатором
	Block	Ідентифікатор користувача	Блокування користувача за заданим ідентифікатором
	Unblock	Ідентифікатор користувача	Розблокування користувача за ідентифікатором
	GetUsers	-	Зчитування усіх користувачів в системі

Таблиця 2.2

Компоненти клієнтського додатку

Назва компоненту	Призначення
add-complaint	Створення нової скарги.
add-proposition	Створення нової пропозиції для виконання дослідження. На основі дослідження створеного власником, формується нова пропозиції, створена потенційним виконавцем.
add-research	Створення нового дослідження. Користувач має ввести заголовок, опис, завантажити прикріплення по необхідності та створити частини.
add-research-review	Створення відгуку про виконання дослідження. Власник дослідження надає відгук про виконавця, вказуючи оцінку (1-5) та вміст.
admin-complaints	Відображення скарг для адміністраторів. Містить всі скарги, які зараз є в додатку і не є закритими.
admin-payments	Проведення платежів для адміністраторів. Реалізує функціонал для переводу коштів між гаманцями користувачів.
admin-users	Відображення всіх користувачів в системі. На даному екрані можна переглянути базову інформацію про користувачів та, при необхідності, заблокувати їх в системі.
complaint	Перегляд скарги. Реалізує функціонал для управління скаргою для адміністраторів. Звичайні користувачі можуть переглядати коментарі та поточний статус виконання скарги.
find-researches	Пошук досліджень, які можна виконати. Фільтрація може бути проведена за категорією, назвою або діапозоном цін.

home	Домашня сторінка користувача. Основною задачою є відображення основної інформації про себе.
login	Сторінка входу в додаток. Для того, щоб перейти на домашню сторінку потрібно ввести коректні ім’я та пароль. Після успішного проходження цього етапу в локальне сховище зберігається JWT-токен, який потім використовується для аутентифікації користувача та авторизації запитів у серверному додатку.
my-complaints	Сторінка для перегляду скарг, створених поточним користувачем.
my-orders	Сторінка для перегляду досліджень, які зараз виконуються користувачем.
my-propositions	Сторінка для перегляду пропозицій, які були зроблені користувачем для певних досліджень.
my-researches	Сторінка для перегляду досліджень, які були створені користувачем.
navbar	Компонент для відображення меню в додатку. Реалізує можливість швидкої навігації по додатку. У пункті платежі міститься поточний баланс користувача.
payment	Сторінка для проведення операцій по поповненню або виведенню коштів з рахунку.
proposition	Сторінка для перегляду пропозиції по дослідженню. Власник дослідження може прийняти її.
register	Сторінка для реєстрації нового користувача.
research	Сторінка для перегляду поточного дослідження. Саме на ній користувач може перейти до частин дослідження, отримати

	загальну інформацію про дослідження. При необхідності, можна скасувати дослідження або написати скаргу. По закінченню виконання є можливість залишити відгук.
research-part	Сторінка для перегляду частини дослідження. На ній користувачі можуть переглянути інформацію про частину дослідження. Також виконавці можуть завантажувати результати виконання частини дослідження, а власники – переглядати їх.
root	Допоміжний компонент для навігації після входу в додаток.
user	Сторінка відображення профілю користувача

ДОДАТОК 3

СИСТЕМА ДЛЯ ЗАХИЩЕНОГО ТА ФРАГМЕНТАРНОГО ВИКОНАННЯ ДОСЛІДЖЕНЬ

Лістинг програмного коду

Аркушів 8

Київ - 2020

ResearchService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Expressions;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using ResearchBuy.Application.Mappers;
using ResearchBuy.Application.Models.Propositions;
using ResearchBuy.Application.Models.Researches;
using ResearchBuy.Application.Models.User;
using ResearchBuy.Application.Providers;
using ResearchBuy.Enums;
using ResearchBuy.Persistence.Contexts;
using ResearchBuy.Persistence.Extensions;
using ResearchBuy.Persistence.Models;
using ResearchBuy.Persistence.Providers;
namespace ResearchBuy.Application.Services.Impl
{
    internal sealed class ResearchService : IResearchService
    {
        private readonly ResearchBuyDbContext _dbContext;
        private readonly IDateTimeProvider _dateTimeProvider;
        private readonly ICurrentUserDataProvider _currentUserDataProvider;
        private readonly IUserMapper _userMapper;
        private readonly IFileMapper _fileMapper;
        private readonly IResearchPartMapper _researchPartMapper;
        public ResearchService(
            ResearchBuyDbContext dbContext,
            IDateTimeProvider dateTimeProvider,
            ICurrentUserDataProvider currentUserDataProvider,
            IUserMapper userMapper,
            IResearchPartMapper researchPartMapper,
            IFileMapper fileMapper)
        {
            _dbContext = dbContext;
            _dateTimeProvider = dateTimeProvider;
            _currentUserDataProvider = currentUserDataProvider;
            _userMapper = userMapper;
            _researchPartMapper = researchPartMapper;
            _fileMapper = fileMapper;
        }

        public async Task Add(ResearchCreationRequest researchCreationRequest)
        {
            var creatorId = _currentUserDataProvider.Current.Id;
            var research = CreateResearch(researchCreationRequest, creatorId);
            await _dbContext.Researches.AddAsync(research);

            for (var i = 0; i < researchCreationRequest.Parts.Count; i++)
            {
                var researchPart = CreateResearchPart(researchCreationRequest.Parts.ElementAt(i), research.Id, creatorId, i);
                await _dbContext.ResearchParts.AddAsync(researchPart);
            }

            await _dbContext.SaveChangesAsync();
        }

        public Task<IReadOnlyCollection<ResearchFindResponse>> GetMyResearches()
        {
            var creatorId = _currentUserDataProvider.Current.Id;

            return _dbContext.Researches
                .Include(x => x.Creator)
                .Include(x => x.ResearchCategory)
                .Where(x => x.CreatorId == creatorId)
                .OrderByDescending(x => x.CreatedAt)
                .Select(CreateResearchFindResponseMappingExpression())
                .ToReadOnlyCollectionAsync();
```

```
    }

    public Task<IReadOnlyCollection<ResearchFindResponse>> GetMyOrders()
    {
        var currentUserId = _currentUserDataProvider.Current.Id;

        return _dbContext.Researches
            .Include(x => x.Creator)
            .Include(x => x.ResearchCategory)
            .Where(x => x.ExecutorId == currentUserId)
            .OrderByDescending(x => x.CreatedAt)
            .Select(CreateResearchFindResponseMappingExpression())
            .ToReadOnlyCollectionAsync();
    }
    private Expression<Func<Research, ResearchFindResponse>> CreateResearchFindResponseMappingExpression()
    {
        return x => new ResearchFindResponse
        {
            Id = x.Id,
            Header = x.Header,
            Category = x.ResearchCategory.Name,
            Status = ResearchStatus.InProgress,
            Price = x.TotalPrice,
            OwnerId = x.CreatorId,
            OwnerFullName = x.Creator.FullName,
        };
    }
    public async Task<IReadOnlyCollection<ResearchFindResponse>> FindResearches(ResearchFindRequest
researchFindRequest)
    {
        var currentUserId = _currentUserDataProvider.Current.Id;
        IQueryable<ResearchPart> query = _dbContext.ResearchParts
            .Include(x => x.Creator)
            .Include(x => x.Research)
            .ThenInclude(x => x.ResearchCategory)
            .Where(x => x.Research.Status == ResearchStatus.Created && x.Research.CreatorId != currentUserId);
        if (!string.IsNullOrWhiteSpace(researchFindRequest.Text))
        {
            var pattern = $"%{researchFindRequest.Text}%";

            query = query.Where(x =>
                EF.Functions.Like(x.Name, pattern) ||
                EF.Functions.Like(x.Description, pattern) ||
                EF.Functions.Like(x.Research.Header, pattern) ||
                EF.Functions.Like(x.Research.Description, pattern));
        }

        if (researchFindRequest.ResearchCategoryId.HasValue)
        {
            query = query.Where(x => x.Research.ResearchCategoryId == researchFindRequest.ResearchCategoryId.Value);
        }

        if (researchFindRequest.MinPrice.HasValue && researchFindRequest.MaxPrice.HasValue)
        {
            query = query.Where(x =>
                x.Research.TotalPrice >= researchFindRequest.MinPrice &&
                x.Research.TotalPrice <= researchFindRequest.MaxPrice);
        }

        var researchParts = await query.ToListAsync();
        return researchParts
            .Select(x => x.Research)
            .GroupBy(x => x.Id)
            .Select(x => x.ToList()[0])
            .Select(x => new ResearchFindResponse
            {
                Id = x.Id,
                Category = x.ResearchCategory.Name,
                Header = x.Header,
```

```
                Status = x.Status,
                Price = x.TotalPrice,
                OwnerId = x.CreatorId,
                OwnerFullName = x.Creator.FullName,
            }).ToList();
    }

    private Research CreateResearch(ResearchCreationRequest researchCreationRequest, int creatorId)
    {
        var researchId = Guid.NewGuid();

        return new Research
        {
            Id = researchId,
            Header = researchCreationRequest.Header,
            Description = researchCreationRequest.Description,
            Status = ResearchStatus.Created,
            CreatorId = creatorId,
            CreatedAt = _dateTimeProvider.Current,
            TotalPrice = researchCreationRequest.Parts.Sum(x => x.Price),
            ResearchCategoryId = researchCreationRequest.ResearchCategoryId,
            ResearchFiles = researchCreationRequest.FileIds.Select(x => new ResearchFile()
            {
                Id = Guid.NewGuid(),
                FileId = x,
                ResearchId = researchId,
            }).ToList(),
        };
    }
    private ResearchPart CreateResearchPart(ResearchPartCreationRequest researchPartCreationRequest, Guid researchId, int
creatorId, int order)
    {
        var researchPartId = Guid.NewGuid();
        return new ResearchPart
        {
            Id = researchPartId,
            Name = researchPartCreationRequest.Name,
            Description = researchPartCreationRequest.Description,
            Status = ResearchPartStatus.Pending,
            ResearchId = researchId,
            CreatorId = creatorId,
            Price = researchPartCreationRequest.Price,
            Order = order,
        };
    }
    public async Task<ResearchQueryResponse> GetResearchById(Guid researchId)
    {
        var anyReview = await _dbContext.ResearchReviews.AnyAsync(x => x.ResearchId == researchId);

        var research = await _dbContext.Researches
            .Include(x => x.ResearchCategory)
            .Include(x => x.Creator)
            .Include(x => x.Executor)
            .Include(x => x.ResearchFiles)
            .ThenInclude(x => x.File)
            .SingleAsync(x => x.Id == researchId);

        var researchParts = await _dbContext.ResearchParts
            .Where(x => x.ResearchId == researchId)
            .ToListAsync();

        var propositionLinks = await GetPropositionLinks(researchId);

        return new ResearchQueryResponse
        {
            Id = research.Id,
            Header = research.Header,
            Description = research.Description,
            TotalPrice = research.TotalPrice,
```

```
        Status = research.Status,
        Category = research.ResearchCategory.Name,
        Creator = _userMapper.Map(research.Creator),
        Executor = _userMapper.Map(research.Executor),
        Files = research.ResearchFiles?.Select(y => _fileMapper.Map(y.File)).ToList(),
        Parts = researchParts.OrderBy(x => x.Order).Select(y => _researchPartMapper.Map(y)).ToList(),
        Propositions = propositionLinks,
        CanLeaveReview = research.Status == ResearchStatus.Completed && !anyReview
    };
}

private async Task<IReadOnlyCollection<PropositionQueryLinkResponse>> GetPropositionLinks(Guid researchId)
{
    var propositions = await _dbContext.Propositions
        .Include(x => x.User)
        .Include(x => x.PropositionResearchParts)
        .ThenInclude(x => x.ResearchPart)
        .Where(x => x.ResearchId == researchId)
        .Select(x => new PropositionQueryLinkResponse
        {
            Id = x.Id,
            ExecutorFullName = x.User.FullName,
            Status = x.Status,
            Price = x.PropositionResearchParts.Sum(y => y.ResearchPart.Price),
        })
        .ToListAsync();
    return propositions;
}
public async Task AddResearchReview(Guid researchId, ResearchReviewCreationRequest researchReviewCreationRequest)
{
    var research = await _dbContext.Researches.SingleAsync(x => x.Id == researchId);
    var researchReview = new ResearchReview
    {
        Id = Guid.NewGuid(),
        Content = researchReviewCreationRequest.Content,
        Mark = researchReviewCreationRequest.Mark,
        ResearchId = researchId,
        ExecutorId = research.ExecutorId.Value,
    };
    await _dbContext.ResearchReviews.AddAsync(researchReview);
    await _dbContext.SaveChangesAsync();
}
public Task<IReadOnlyCollection<ResearchReviewResponse>> GetReviews()
{
    var currentUserId = _currentUserDataProvider.Current.Id;
    return _dbContext.ResearchReviews
        .Where(x => x.ExecutorId == currentUserId)
        .OrderByDescending(x => x.CreatedAt)
        .Select(x => new ResearchReviewResponse
        {
            Id = x.Id,
            Content = x.Content,
            Mark = x.Mark,
            CreatedAt = x.CreatedAt,
            User = new UserLinkResponse
            {
                Id = x.Executor.Id,
                FullName = x.Executor.FullName,
            }
        })
        .ToReadOnlyCollectionAsync();
}

public async Task<ResearchStatusResponse> GetStatus(Guid researchId)
{
    var status = await _dbContext.Researches
        .Where(x => x.Id == researchId).Select(x => x.Status)
        .SingleAsync();
```

```
            return new ResearchStatusResponse
            {
                Status = status,
            };
        }

        public async Task<ResearchFailResponse> Fail(Guid researchId)
        {
            var research = await _dbContext.Researches.SingleAsync(x => x.Id == researchId);
            var researchParts = await _dbContext.ResearchParts.Where(x => x.ResearchId == researchId).ToListAsync();
            if (researchParts.Any(x => x.Status == ResearchPartStatus.OnReview))
            {
                return new ResearchFailResponse
                {
                    IsSucceeded = false,
                    Message =
                        "Sorry, but you can't fail the research while there is an part on review. Please, contact support to solve this issue",
                };
            }

            research.Status = ResearchStatus.Failed;
            _dbContext.Researches.Update(research);

            foreach (var part in researchParts.Where(
                x => x.Status == ResearchPartStatus.InProgress || x.Status == ResearchPartStatus.Pending).ToList())
            {
                part.Status = ResearchPartStatus.Failed;
                _dbContext.ResearchParts.Update(part);
            }
            await _dbContext.SaveChangesAsync();
            return new ResearchFailResponse
            {
                IsSucceeded = true,
                Message = "The research is finished. Please, write the review on the research.",
            };
        }
    }
}
```

ResearchPartService.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Http;
using Microsoft.EntityFrameworkCore;
using ResearchBuy.Application.Models.Files;
using ResearchBuy.Application.Models.Payments;
using ResearchBuy.Application.Models.Researches;
using ResearchBuy.Application.Providers;
using ResearchBuy.Enums;
using ResearchBuy.Persistence.Contexts;
using ResearchBuy.Security;
using ResearchBuy.Security.DataStructures;
using ResearchBuy.Security.Models;
namespace ResearchBuy.Application.Services.Impl
{
    internal sealed class ResearchPartService : IResearchPartService
    {
        private readonly ResearchBuyDbContext _researchBuyDbContext;
        private readonly IBlockchainService _blockchainService;
        private readonly IResearchPartCommentService _researchPartCommentService;
        private readonly ICurrentUserDataProvider _currentUserDataProvider;
        private readonly IPaymentService _paymentService;

        public ResearchPartService(
            ResearchBuyDbContext researchBuyDbContext,
            IBlockchainService blockchainService,
```

```
        IResearchPartCommentService researchPartCommentService,
        ICurrentUserDataProvider currentUserDataProvider,
        IPaymentService paymentService)
    {
        _researchBuyDbContext = researchBuyDbContext;
        _blockchainService = blockchainService;
        _researchPartCommentService = researchPartCommentService;
        _currentUserDataProvider = currentUserDataProvider;
        _paymentService = paymentService;
    }

    public async Task<ResearchPartFullQueryResponse> GetById(Guid researchPartId)
    {
        var researchPart = await _researchBuyDbContext.ResearchParts
            .Include(x => x.Research)
            .SingleAsync(x => x.Id == researchPartId);

        var reviewResultCheck = await CanUserCheckTheResultsOfTheResearchPart(researchPartId);

        var blockchain = reviewResultCheck.CanViewResults
            ? await _blockchainService.GetBlockchain(researchPart.Research.LastCompletionPartHash) : null;

        var comments = await _researchPartCommentService.GetComments(researchPartId);

        return new ResearchPartFullQueryResponse
        {
            Id = researchPart.Id,
            Name = researchPart.Name,
            Description = researchPart.Description,
            Price = researchPart.Price,
            Status = researchPart.Status,
            ResearchId = researchPart.ResearchId.Value,
            CreatorId = researchPart.Research.CreatorId,
            ExecutorId = researchPart.Research.ExecutorId,
            ResearchHeader = researchPart.Research.Header,
            Comments = comments,
            CanViewResearchResults = reviewResultCheck.CanViewResults,
            ResultsViewErrorMessage = reviewResultCheck.Message,
            Results = blockchain?.Blocks
                .Where(x => x.Data.ResearchPartId == researchPartId)
                .Select(x => new ResearchPartResultSubmissionResponse
                {
                    ResearchPartResultId = x.Data.ResearchPartResultId,
                    Description = x.Data.Description,
                    IsAccepted = x.Data.IsAccepted,
                    Files = x.Data.Files.Select(x => new FileResponse
                    {
                        Id = x.Id,
                        FileName = x.Name,
                    }).ToList(),
                }).ToList(),
        };
    }
    private async Task<(bool CanViewResults, string Message)> CanUserCheckTheResultsOfTheResearchPart(Guid
researchPartId)
    {
        var user =
            await _researchBuyDbContext.Users.SingleAsync(x => x.Id == _currentUserDataProvider.Current.Id);
        var researchPart = await _researchBuyDbContext.ResearchParts
            .Include(x => x.Research)
            .SingleAsync(x => x.Id == researchPartId);
        if (researchPart.Research.CreatorId != _currentUserDataProvider.Current.Id || researchPart.Status !=
ResearchPartStatus.OnReview)
            return (true, string.Empty);

        var totalPrice = await _researchBuyDbContext.ResearchParts
            .Where(x => x.CreatorId == _currentUserDataProvider.Current.Id &&
                        x.Status == ResearchPartStatus.OnReview)
            .SumAsync(x => x.Price);
```

```
            var canReviewResults = user.Balance >= totalPrice;
            var message = canReviewResults
                ? string.Empty
                : $"Account balance on the account must be bigger or equal to the sum of the prices of research parts in the on review state. Please, deposit {totalPrice - user.Balance}$.";

            return (canReviewResults, message);
        }

        public async Task SubmitPartExecutionResult(Guid researchPartId, string description, IReadOnlyCollection<IFormFile> attachedFiles)
        {
            var researchPart = await _researchBuyDbContext
                .ResearchParts
                .Include(x => x.Research)
                .SingleAsync(x => x.Id == researchPartId);

            var lastCompletionPartHash = researchPart.Research.LastCompletionPartHash;

            var researchPartCompletionResult = new ResearchPartResult
            {
                ResearchPartResultId = Guid.NewGuid(),
                ResearchPartId = researchPartId,
                Description = description,
                IsAccepted = false,
                Files = await MapManyAsync(attachedFiles, MapFormFileToResearchPartFile),
            };

            lastCompletionPartHash = await _blockchainService.AddBlock(lastCompletionPartHash, researchPartCompletionResult);

            var research = await _researchBuyDbContext.Researches.SingleAsync(x => x.Id == researchPart.ResearchId);
            research.LastCompletionPartHash = lastCompletionPartHash;

            researchPart.Status = ResearchPartStatus.OnReview;
            _researchBuyDbContext.Researches.Update(research);
            _researchBuyDbContext.ResearchParts.Update(researchPart);
            await _researchBuyDbContext.SaveChangesAsync();
        }
        public async Task<FileContentResponse> GetFileContent(Guid researchPartId, Guid researchPartResultId, Guid fileId)
        {
            var researchPart = await _researchBuyDbContext.ResearchParts
                .Include(x => x.Research)
                .SingleAsync(x => x.Id == researchPartId);
            var hash = researchPart.Research.LastCompletionPartHash;
            var blockchain = await _blockchainService.GetBlockchain(hash);
            var block = blockchain.Blocks.Single(x => x.Data.ResearchPartResultId == researchPartResultId);
            var file = block.Data.Files.Single(x => x.Id == fileId);
            return new FileContentResponse
            {
                FileName = file.Name,
                Content = file.Content,
            };
        }
        public async Task AcceptResultOfPart(Guid researchPartId, Guid researchPartResultId)
        {
            var researchPart = await _researchBuyDbContext.ResearchParts
                .Include(x => x.Research)
                .SingleAsync(x => x.Id == researchPartId);

            var research = researchPart.Research;
            var blockchain = await _blockchainService.GetBlockchain(research.LastCompletionPartHash);
            var acceptedPart = blockchain.Blocks.Single(x => x.Data.ResearchPartResultId == researchPartResultId);
            acceptedPart.Data.IsAccepted = true;
            var partsToBeUpdated = blockchain.Blocks
                .SkipWhile(x => x != acceptedPart)
                .ToList();
            var currentHashes = blockchain.GetCurrentHashes(research.LastCompletionPartHash);
            var previousHash = acceptedPart.PreviousHash;
```

```
            foreach (var part in partsToBeUpdated)
            {
                var oldHash = currentHashes[part];
                part.PreviousHash = previousHash;
                previousHash = await _blockchainService.EditBlock(oldHash, part);
            }

            research.LastCompletionPartHash = previousHash;
            researchPart.Status = ResearchPartStatus.Finished;

            _researchBuyDbContext.Update(research);
            _researchBuyDbContext.Update(researchPart);

            await _researchBuyDbContext.SaveChangesAsync();
            await UpdateResearchStatusIfAllPartsAreCompleted(research.Id);
            await _paymentService.UnblockFunds(researchPart.Price, research.CreatorId);
            var transferRequest = new TransferRequest
            {
                Amount = researchPart.Price,
                FromUserId = researchPart.CreatorId,
                ToUserId = research.ExecutorId.Value,
                BothFromAvailableAndBalance = true,
            };

            await _paymentService.Transfer(transferRequest);
        }

        private async Task UpdateResearchStatusIfAllPartsAreCompleted(Guid researchId)
        {
            var allFinished = await _researchBuyDbContext.ResearchParts
                .Where(x => x.ResearchId == researchId)
                .AllAsync(x => x.Status == ResearchPartStatus.Finished);

            if (!allFinished)
                return;

            var research = await _researchBuyDbContext.Researches.SingleAsync(x => x.Id == researchId);
            research.Status = ResearchStatus.Completed;
            _researchBuyDbContext.Researches.Update(research);
            await _researchBuyDbContext.SaveChangesAsync();
        }
        public async Task UpdateStatus(Guid researchPartId, ResearchPartStatusUpdateRequest request)
        {
            var researchPart = await _researchBuyDbContext.ResearchParts.SingleAsync(x => x.Id == researchPartId);
            researchPart.Status = request.Status;
            _researchBuyDbContext.Update(researchPart);
            await _researchBuyDbContext.SaveChangesAsync();
        }
        private async Task<IReadOnlyCollection<TOutput>> MapManyAsync<TInput, TOutput>(IReadOnlyCollection<TInput>
input, Func<TInput, Task<TOutput>> map)
        {
            var inputTasks = input.Select(map).ToList();
            await Task.WhenAll(inputTasks);
            return inputTasks.Select(x => x.Result).ToList();
        }
        private static async Task<ResearchPartResultFile> MapFormFileToResearchPartFile(IFormFile formFile)
        {
            await using var memoryStream = new MemoryStream();
            await formFile.CopyToAsync(memoryStream);
            return new ResearchPartResultFile
            {
                Id = Guid.NewGuid(),
                Name = formFile.FileName,
                Content = memoryStream.ToArray()
            };
        }
    }
}
```